The CIRIS Constitution: The Wire

A volume of edition 1.0-rc10

Stewarded by Eric Moore

The Wire

What this is. The specification an implementer builds a second, independent producer and consumer from: the envelope and its members (2.1), the closed set of five row types (2.4), the signed object and its byte vectors (2.6.1.7), canonicalisation and its determinism rules (2.6), identifiers and time (2.6.2 to 2.6.8), and the transport, custody, key and privacy rules of Part 5 with the adversary model of 1.13.3.

Whom it binds. Any CEG-Conforming Producer, Consumer or Substrate. Conformance is judged against this volume, *The Dictionary* and the reader's policies, and against nothing else.

What it promises. One envelope, one signing input, one signature construction; a closed row-type slot; byte identity across implementations as a target with a published vector set as its proof; a refusal with a named reason, never a silent truncation; a fixed-size substrate envelope and a 1 MiB cap on the signed object.

What it leaves out. The meaning of any dimension, which is the dictionary's; the reasons a rule exists, which are the charter's; and whether a given MUST has shipped, which is the register's. Where this volume states a rule as normative, it means the bytes; where it explains, it says so.

Where the rest lives. The generated artefacts this volume is the source of: `manifests/signed_object.cddl`, `manifests/signed_object_vectors.json`, and `manifests/namespace_registry.json` with its match vectors. An implementer should read those beside the text, because the text is checked against them in CI and they are what a second implementation is measured by.

Part 1 — Foundation

1.13.3 Adversary model & privacy non-goals (normative)

Fidelity demands the format not overclaim what it protects. CEG makes confidentiality and integrity claims; this section bounds them. **The word "privacy" in this spec means exactly two things and no more: (1) content-holding confidentiality and (2) cohort-scoped visibility.** It does **not** mean metadata privacy, communication-graph privacy, or unobservability. Implementers and operators MUST NOT represent CEG as providing the stronger properties.

1.13.3.1 Non-goals: what omission does NOT buy

The following are **explicitly out of scope** at the base CEG/RET layer; treating them as provided is an error:

- **Relationship-existence privacy.** The *existence* of a self-collective / family / community and its membership-change events are observable: `family_id` / `community_id` ride the envelope, and admission / removal / consensus-protocol changes emit `hard_case:*` reserved-prefix events (CC 3.4.4–3.4.2) into the log. An observer learns that a group exists, roughly how big it is, and when its membership churns.
- **Communication-graph / metadata privacy.** DNS-free member resolution (CC 4.4.3.2.4.1) plus Reticulum announce / path-request expose *who is reachable where*; the federation directory + `transport_destination` bindings name endpoints. A passive network observer or an honest-but-curious member can reconstruct a substantial portion of the **who-talks-to-whom** graph. Cohort scope hides *content*, not *contact*.
- **Traffic-analysis resistance.** Encrypted streams still leak via side channels the wire format does not pad or cover: the CC 5.3.3.3 STH cadence (default T=2 s), the churn-driven key-cascade volume/timing (CC 5.1), and per-chunk size/rate. An observer can infer stream existence, approximate group size, churn rate, activity bursts, and often media bitrate class — without decrypting a byte.
- **Unobservability against a *global passive adversary*. **Base CEG/RET does not, on its own, defeat an adversary who observes every network link. At** federation (public-commons) scope** the transport reveals path endpoints and traffic patterns, so full sender/receiver unobservability *at that scope* is a **separate, opt-in** mechanism (the CIRISNodeCore Anonymous Tier — Sphinx onion routing). This is the residual non-goal. It does **not** mean CEG provides no anonymity: at every cohort scope below federation, structural anonymity *to outsiders* is the default — see CC 1.13.3.4.
- **Post-compromise security (PCS) for streams.** The CC 4.5.12.1 Option-A choice is forward-only: a member removed at epoch *e* cannot read epoch *e+1*, but a *compromised current member's* key is not self-healed by a key-update the way MLS PCS provides.

1.13.3.2 What the structural-invisibility primitive (CC 5.2) buys

Keeping every holding claim inside its own cohort (CC 5.2) gives `cohort_scope: self | family` content **content-holding confidentiality**: a non-member cannot *discover that the bytes exist via the substrate* and cannot *fetch* them (no holder is advertised; the bytes are delivered

only to admitted members via the at-rest key cascade). End-to-end content confidentiality is additionally provided by the per-epoch DEK (hybrid X25519+ML-KEM-768) and AES-256-GCM. That is the whole of what omission buys.

1.13.3.3 Adversary classes (and where each is / is not addressed)

Adversary	**Addressed**	**NOT addressed**
Passive network observer	content confidentiality (AEAD + DEK); equivocation (STH)	comm-graph, traffic analysis, group size/churn inference
Honest-but-curious member	— (members see in-scope content by design)	can enumerate co-members + reconstruct local comm-graph
Malicious member	cannot forge others' attestations; removal is forward-secret	can leak content they were entitled to; metadata as above
Compromised substrate node	cannot decrypt self/family content (no DEK); CEG-native replication carries signed provenance	can observe directory metadata + traffic patterns it routes
Equivocating producer	**mitigated** — per-stream STH (CC 5.3.3.3) + consistency proofs (CC 5.3.1.1): cannot show different chunk-K to different viewers nor rewrite midstream	—
Equivocating founder (trust-root lineage)	**mitigated** — witnessed lineage heads (CC 3.2 T6, CC 5.3.1): two witnessed heads for one lineage are `hard_case: lineage_equivocation`, never a silent first-seen fork; exposure bounded by the charter's `witness_cadence`; a stale prefix cannot be attached to (CC 3.2 T4a)	a founder set that colludes with every witness of its own lineage; an attached node's view between cadence ticks
Placeholder identity (a registered key with no live agent behind it — the ERC-8004 field finding that 3 %, 4 % and 15 % of registrations on three chains exposed a valid registration file with a live endpoint; Xiong, X., Li, Z., Wei, W., Wang, Q., Knottenbelt, W., & Wang, Z. (2026), "Can Trustless Agents Be Trusted? An Empirical Study of the ERC-8004 Decentralized AI Agent Ecosystem," arXiv: 2606.26028)	**addressed by construction** — CEG keeps no registry whose entry constitutes agent-hood: a key is an identifier, and standing comes only from attestations others make about it under policies the consumer chose (CC 2.3.2, CC 4.4.3.8 Policy A); an `agent` or `node` key MUST have a bound human steward, a `user` identity it is an admitted `identity_occurrence` of, to hold non-infrastructure membership (CC 3.2 steward-binding gate); liveness is never a durable claim — entitlement is attested, reachability is an ephemeral node-local fact that is never an attestation and never logged (CC 5.3.3.4 D6), so presence in any directory never reads as "live"	minting a key costs nothing; the CC defines no discovery-time liveness predicate beyond Edge reachability at query time, so discovery tooling that hides unreachable identities is client policy; a self-attested `identity_occurrence` is self-assertion until cross-attested (CC 3.1.1)

Adversary	Addressed	NOT addressed
Coordinated reviewer capture (Sybil feedback — 73.6 %, 59.2 % and 90.6 % of ERC-8004 reviewers on three chains flagged as coordinated, and a median agent's reputation moved past threshold by one feedback at a median cost of $0.055, $0.0042 and $0.0027; same study)	**mitigated** — influence is gated by trust, not by registration: the conformance-floor policy composes only attesters in the consumer's pinned trust set (CC 4.4.3.8 Policy A), so a fresh key has no weight until someone the consumer trusts confers it, and the richer policies (CC 4.4.3) root in admitted membership; detector dimensions aggregate by median (CC 4.4.2); on the aggregation leg, correlated sources count as one at full correlation (the correlated-source discount, CC 6.1.2.1.2) — mechanized in coherence-ratchet as a consequence of the Kish definition and not yet measured on any attester population (grades at CC 8.6.1); RATCHET's own lake proves the identity's K1–K4 properties; `ratchet:flag:coordinated_voting_cluster` names the pattern (CC 3.1.6)	the default aggregation for `signed` dimensions is a plain mean of `score` × `confidence` inside the trust set (CC 4.4.2) with no bounded-influence rule, so one captured trusted attester moves it (named as a bet, CC 8.3.1 R14); `evidence_refs` are required only where a dimension's rules say so, so an attestation with no verifiable interaction behind it is admissible at the grammar level and filtered only by consumer policy; `ratchet:flag:*` is advisory and carries no evidence row
Census stuffer (forged census returns move a relief, a default, a cap or a vote base)	**bounded** — a valid return adds exactly one count to one bucket per field (the one-hot proof, CC 6.1.9); per-node composites publish the median bucket, which moves at most one bucket below a fifth of returns forged; voice counts are owner-bound, and the dominance gate refuses a composite one owner dominates	an adversary holding a real share of rooted owners moves counts by that share (the admission cost, not a census property); a forger that returns plausible values is not detected — the census bounds influence, it does not detect lies
Census deanonymiser (small cells, sparse regions, differencing consecutive composites)	**mitigated** — ten owners per published cell, continent-sized regions, levels never deltas, values rounded to a grid, bucketed returns, and no aggregator sees a return at all	an observer who controls all but one owner in a cell infers the last one (nine colluding owners in that cell)
Compelled aggregator or directory (subpoena, coerced operator)	**mitigated** — an aggregator holds uniformly random shares and a public partial; a return's value is exposed only if both aggregators of a period are compelled or collude; a share is deleted once summed; the directory holds no census data	the aggregator sees which node sent it a share in the period (participation, which is everyone's)
Global passive observer on census timing	**mitigated** — one fixed-size return per period at an offset from `H(return_token ‖ period)`; on a shaped link (CC 5.4.3) it replaces a cover frame	on an unshaped link the observer sees that a node participates, which every eligible node does
Withholding adversary (suppresses returns to shrink a count or a vote base)	**bounded** — the base falls at most a tenth per period and never below its floor; a fall of more than a third freezes the base and every narrowing relief until two stable periods follow	a sustained silence lowers the base to its floor over many periods, in plain view

Adversary	Addressed	NOT addressed
Poisoned aggregator (misstates, drops or invents returns)	**mitigated** — two aggregators, each partial checked against the other's commitment and share count, buckets against `source_count`, a contributor's inclusion path against the commitment; only serve seats or community members aggregate, and they are accountable	two colluding aggregators can drop returns or invent tokens up to their expected share, bounded to the rounding grid by the directory's count of live nodes; the per-seat load check is statistical until the seat order is a signed row

Operator guidance: cohort-scoped confidentiality and anonymity-to-outsiders are on by **default** (CC 1.13.3.4). If a deployment *additionally* requires unobservability against a **global passive adversary** at federation scope (e.g., under a totalitarian-threat model), it MUST layer the Anonymous Tier; base CEG/RET is not sufficient for that strongest model. **Using** the Anonymous Tier is the publisher's opt-in; **relaying** it for others is the default: every node relays onion traffic unless it opts out, and a serve node cannot opt out, so that running a relay identifies no one and every operator can say only that it runs the standard substrate (CC 5.4.6). Its first scope is federation-scope publication. State both the default protection and its residual limit in any user-facing privacy representation.

1.13.3.4 Anonymity defaults to the smallest cohort scope (normative)

Anonymity defaults to the smallest cohort scope consistent with a publication's intent; **federation (public-commons) scope is the opt-in**, not anonymity. This is the protective default, chosen over opt-in anonymity: under opt-in, the non-savvy vulnerable — an abuse victim, a teenager in a hostile household — fail to obtain a protection that a motivated adversary obtains trivially, so opt-in weakens protection only for those least able to navigate it. CEG therefore makes cohort-scoped confidentiality and anonymity toward outsiders the *default* — anonymity here means outsiders cannot see a cohort's content or map its addresses to it; it never means an unattributed sender, since every link is attributed to a key and a first contact is unowned but never anonymous (CC 3.2 (e), CC 5.3.3.5 E1) — content born and living at `self` / `family` / `community` is never advertised to outsiders (the structural-invisibility primitive of CC 1.13.3.2) — where, for `self`, **"outsider" means any node not owner-bound to my owner**: the `self` cohort is one owner's own devices, so its boundary is well-defined only because ownership is **single-valued** (CC 3.2 single-owner invariant) — and reserves opt-in only for the strongest threat model: full unobservability against a global passive adversary at federation scope (CC 1.13.3.1).

CEG declines client-side scanning — it would be the surveillance backdoor this framework refuses — and does **not** claim to detect harmful content inside private cohorts; that is the same wall every end-to-end-encrypted system hits, and the document does not paper over it. What CEG adds beyond systems that ship default privacy with no in-network governance is **fails-secure governance** (the named-moderator existence invariant, CC 4.5.4) and a **substrate-protective perceptual-hash tripwire at every scope-*widening* promotion event** — a promotion into the Commons, or a post into a `public` room (CC 3.2), creates the Contribution that is hash-matched at that seam — never on the device, never inside a private cohort. This recovers the spreading-correlated child-safety catches (the harm vector that creates new victims) without scanning content that stays within its cohort. The honest line: fails-secure governance plus accountable, censorship-resistant moderation; never private-content detection.

The opt-in boundary is drawn at GPA-unobservability because it is the one protection that is gen-

uinely costly (onion-routing latency, mixing, dedicated cover) *and* is only reached by an operator already taking the deliberate step of publishing to the public commons — so the differential-uptake argument that makes cohort anonymity a *default* does not reach it. **This boundary is provisional:** as the actual overhead of GPA-resistance is measured, the line MAY move toward making more of it default (e.g., for individual federation-scope publishers) if the cost proves low enough to absorb.

Part 2 — The Grammar

2.1 The envelope

The envelope is where a claim becomes accountable. Every `scores` Attestation carries it, and every field below either binds the claim to evidence, names who may revoke it, or records the conditions under which it was made — the integrity guarantees that let a consumer trust a stranger's signed word. Field semantics are consolidated here once; the rest of the Part references back to this table.

Field	Required	Description
`attesting_key_id`	(substrate field)	Attester's `federation_keys.key_id`.
`attested_key_id`	(substrate field)	Subject's `federation_keys.key_id`.
`dimension`	yes	The canonical namespace prefix + scoped leaf. Persist treats this as TEXT; consumers parse against CC 3.1's namespace map.
`score`	yes	Pos/neg scalar in [-1, +1]. Polarity is encoded by sign; magnitude carries strength. Some dimensions are boolean-via-score (±1 only); some are positive-only; some are signed; per-dimension table in CC 3.1 names the polarity.
`confidence`	yes	The attester's own confidence in their score. [0, 1]. Low confidence + high magnitude = "I believe this strongly but I might be wrong"; high confidence + low magnitude = "I am sure the truth is near-neutral."
`context`	no	Free-form scoping detail. Not parsed by the substrate; used by consumers + audit + RATCHET.
`evidence_refs`	no (often required by per-dimension policy)	List of URIs / content-hashes pointing to backing evidence (Stripe receipt, licensing-body record, observed interaction, log entry, audit-chain leaf, etc.). Some dimensions in CC 3.1 require non-empty evidence_refs.
`valid_until`	no	ISO 8601 datetime per CC 2.6.2. If set, consumer policy treats the attestation as stale after that point (independent of the substrate row's own `expires_at`).
`epistemic_mode`	no	Per CC 2.5 Epistemic-mode axis; default `direct`. Consumers may weight by mode (e.g., direct witness > hearsay).
`witness_relation`	no	`self` \
`oversight_mode`	no	`HITL` \
`occurrence_id`	no	Identifies which occurrence of a multi-occurrence agent deployment emitted this attestation. Format: `"occurrence-{n}"` per the agent's `AGENT_OCCURRENCE_ID` env var, or `"__shared__"` for shared-task pattern emissions. Default `null` → treated as `occurrence-0` for backward compat. **Self-asserted**: this field is NOT cryptographically bound to a fleet-attestation primitive; an adversary running a single key can claim any occurrence_id. Acknowledged design tradeoff per CC 8.3.1.

Field	Required	Description
occurrence_count	no	Total occurrences in the deployment fleet emitting the attestation; integer ≥ 1. Default null → 1 (single-occurrence). Same self-assertion caveat as occurrence_id.
occurrence_role	no	primary \
stake	no	Per CC 2.5 Stake axis; default reputational. Composes with the attester's actual stake-backed-by attestations from CC 3.1.1.
community_id	no (REQUIRED iff cohort_scope ∈ {community, affiliations})	The community_key_id of the community this Contribution is scoped to, per CC 3.2 community subject_kind. One identity MAY belong to multiple communities; the field disambiguates which community's roster gates visibility. Required iff cohort_scope ∈ {community, affiliations} — an affiliation is a community (CC 4.4.3.2.8), and a hierarchy edge resolves its role_term through this member; the substrate rejects community- or affiliation-scoped Contributions missing the field. Parallel to family_id but with different semantics: community content is encrypted at rest under the per-community DEK, and its holding claims (custody:ack:v1, CC 3.1.3.3) reach the community's members only — structural invisibility (CC 5.2) holds at every bounded scope.
family_id	no (REQUIRED iff cohort_scope == family)	The family_key_id of the family this Contribution is scoped to, per CC 3.3.4 family subject_kind. One identity MAY belong to multiple families (CC 4.4.3.4 Policy L); the field disambiguates which family's DEK applies and which membership roster gates visibility. Required iff cohort_scope == family (substrate rejects family-scoped Contributions missing the field). Composes with CC 5.2 structural-invisibility — no holding claim for 'cohort_scope: self \
session_id	no (REQUIRED on session:*)	The exchange a session:* claim is about; with community_id it is the claim's key (CC 3.1.3.1). Verbatim, case-preserved.
claimed_at	no (REQUIRED on session:*)	When the claiming occurrence took the session, in the CC 2.6.2 canonical form; the convergent merge orders on it (CC 3.1.3.1).
scope	no (REQUIRED on consent:state:granted)	What the row covers: a bare token or an array of tokens. On a consent:state:granted row it is **the grant's scope** (CC 3.3.1: retain · share · analyze · train · publish, sub-scoped in the token as retain:90d; a grant naming no scope matches nothing); on a delegates_to it is the delegated scope set (CC 4.4.3.4.3). Inside the signed bytes like every member; a consumer that does not know the member preserves it per CC 2.1.1 and MUST NOT compose a capacity:* verdict that ignores it.
custody_state	no (REQUIRED on custody:ack)	here \
group_kind	no (REQUIRED on membership:proposal)	family \

Field	Required	Description
`role`	no (nullable on `membership:proposal`; restated on `membership:acceptance`)	The role the proposal offers the invitee; an acceptance restates it, and a widening that assigns a different role does not match (CC 3.1.3.2).
`expires_at`	no (REQUIRED on `membership:proposal`)	The proposal's signed expiry, CC 2.6.2 canonical form, after its `asserted_at` and at most 30 days later; the acceptance and the widening are each judged against it on their own signed instants, never a receiver's clock (CC 3.1.3.2).
`proposal_hash`	no (REQUIRED on `membership:acceptance` and `membership:decline`)	The answered proposal's `original_content_hash`; with `references_attestation_id` it binds an answer to exactly one proposal (CC 3.1.3.2).
`additional_scrubs`	no	**Quorum on a row** (CC 2.4 cosigned rows): a list of hybrid signatures (Ed25519 + ML-DSA-65) over the same canonical bytes as the row's own, each `{scrub_key_id, scrub_signature_classical, scrub_signature_pqc, cosigned_at?}` (CC 2.6.1.7). It travels **beside** the signed object, never inside it, since a cosignature cannot be inside the bytes it signs; each cosigner's role is read from the receiver's fold, not from the entry. This is the quorum carrier; the name `cosignatures[]` is retired. The family declares which `consensus_protocol` the cosigners must satisfy; a cosigned row is retracted only by a cosigned `withdraws` (CC 2.4.1.1). The general form of the ceremony plane's scrub set (CC 3.2 T2).
`subject_sha256_ed25519_raw`	no	On a `moderation:*` score: the subject key named by the SHA-256 of its raw 32-byte Ed25519 public key, 64 lowercase hex per CC 2.6.3, so the row does not carry the key's cleartext label (CC 2.4.1.1). A malformed value is refused at admission.
`attached_head_digest`	no (REQUIRED on `trust:accepts:v1`)	The SHA-256, lowercase hex per CC 2.6.3, of the lineage head the acceptance edge attaches on — the signed roster record at that version (CC 3.2 T4a). Checked at the edge's **first admission only** against the head's witness cosignatures and the root's `attach_window_secs`; never re-checked on a node's own restore. An edge whose digest is later un-headed by a lowered `revoked_after` is read as naming that head's nearest verifying ancestor (T8 viii).
`attach_window_secs`	no (on `trust:charter:v1`)	Integer seconds. The oldest a lineage head's newest counted witness cosignature may be for a consumer to *attach* to this root (CC 3.2 T4a); MUST exceed `witness_cadence_secs` plus the CC 2.6.7 skew tolerance. A charter that declares none makes its root attachable only through an out-of-band anchor to a specific head (T5). Never a validity leg for an attached root (T4).
`witness_cadence_secs`	no (REQUIRED on the `trust:charter:v1` of a witnessed lineage)	Integer seconds. How often each directory witness MUST re-cosign the current head, unchanged; the equivocation-exposure window (CC 3.2 T6).

Field	Required	Description
`halt_fuse_secs`	no (on the accord's `trust:charter:v1`)	Integer seconds. How long a `constitutional` halt signed by ONE holder stands before it lapses unless a strict majority of the standing roster confirms it (CC 4.2.1. 1). **Absent means 86400**: the value is a constant of this document, read by the verifier, not a substrate-internal default, so a charter minted without the member is conformant and carries the 24-hour fuse; an explicit value overrides it, by re-scrub.
`recovery_commitments`	no (REQUIRED on the accord's `trust:charter:v1`, CC 4.2.6)	Object `{holder_key_id: commitment}`; each commitment is lowercase-hex SHA-256 over the JCS array of the committed recovery key's `{key_id, pubkey_ed25519_base64, pubkey_ml_dsa_65_base64}` exactly as stored on its key record (CC 3.2 T3), a one-element array here. It binds key material, never only the id. A holder rotates lost signing-key material by a self-`supersedes` under a key whose record hashes to its entry.
`witness_quorum`	no (REQUIRED on the `trust:charter:v1` of a witnessed lineage)	Integer. Cosignatures from distinct independent witnesses required for a head to count as witnessed; MUST equal ⌊
`witnesses`	no (REQUIRED on the `trust:charter:v1` of a witnessed lineage)	Array of `SignedKeyRecord`: the lineage's witness directory (CC 3.2 T6), inside the charter's scrub-signed bytes and therefore inside the genesis bundle for the shipped roots (CC 5.3.4). Each MUST have `identity_type` ⊇ `{witness}` and be independent of the lineage (not a founder, not a member seat, not owner- or steward-bound to a founder). A change is a charter re-scrub, witnessed by the directory before it.
`consent_supersedes`	no (on `consent:state:*`)	`uuid`. The prior consent-state row this one supersedes (a renewal, or the substrate's `consent:state:expired` naming the grant that lapsed, CC 3.3.1).
`deletion_window`	no	`tstr`. The deadline by which a consented payload must be deleted once consent ends, carried on the grant (CC 3.3.1); absent means the family's default.
`widened_at`	no (REQUIRED on a widening `supersedes`)	RFC 3339 with milliseconds (CC 2.6.7). The instant a widening took effect; it is the `scrub_timestamp` of the widening's signature (CC 4.4.3.3.1).
`accord_root`	no (REQUIRED on `accord:*` rows)	`key_id`. Which accord an `accord:*` row belongs to (CC 4.2).
`consent_basis`	no (carried on `capacity:*` and `observation:access_assumption:v1` rows)	`[* {principal: key_id, paths: [* "subject" / "occurrence_of" / "delegates_to"], attestation_id: uuid}]`. The consent grants a witness relied on: for each principal, the paths by which its grant reached the subject and the `attestation_id` of the `consent:state:granted` row. Validated at the family's gate.
`n_eff_pr`	no (carried on `capacity:*` rows)	`number`. The effective sample size of the scoring window as a participation ratio of the feature covariance spectrum; the gate reads this one.
`n_eff_h`	no (carried on `capacity:*` rows)	`number`. The same window's effective sample size as an entropy perplexity, `exp(-Σ p·ln p)` over the normalised spectrum; carried for audit, not gated.
`sample_size`	no (carried on `capacity:*` rows)	`uint`. Rows in the scoring matrix.
`feature_dim`	no (carried on `capacity:*` rows)	`uint`. Features per row in the scoring matrix.
`sample_size_gate`	no (carried on `capacity:*` rows)	`uint`. The configured minimum sample size the score was gated on.

Field	Required	Description
target_n_eff	no (carried on capacity:* rows)	number. The configured target effective sample size.
min_featur e_dim	no (carried on capacity:* rows)	uint. The configured minimum feature dimension.
n_eff_floor	no (carried on capacity:* rows)	number. The configured effective-sample-size floor below which no score is emitted.
window	no (carried on capacity:* and observation:access_a ssumption:v1 rows)	uint. The scoring or observation window, in rows.
trace_level	no (REQUIRED on observation:access_a ssumption:v1)	generic \
occurrence _key_ids	no (REQUIRED on observation:access_a ssumption:v1)	[* key_id], a set. The occurrences whose rows the witness read.
window_from	no (REQUIRED on observation:access_a ssumption:v1)	RFC 3339 with milliseconds (CC 2.6.7). Start of the observation window.
window_to	no (REQUIRED on observation:access_a ssumption:v1)	RFC 3339 with milliseconds. End of the observation window.
amendment_ digest	no (REQUIRED on identity_amendment:v 1)	hex64. SHA-256 of the identity amendment as applied (CC 3.1.5).
authorized_by	no (REQUIRED on identity_amendment:v 1)	uuid. The attestation_id of the deferral:r esponse:approved:v1 row that authorised the amendment.
channel_cl asses	no (REQUIRED on observation:access_a ssumption:v1)	[* tstr], a set. The classes of channel through which the witness saw the subject, as the emitter names them.
subject_ke y_ids	no	List of consent-holder key_ids for this Contribution. Each entry MAY be a federation_keys.key_id OR a canonical-hash identifier. Each listed key has substrate-recognized authority to (a) issue withdraws against this Contribution and (b) emit consent:* dimensions about this Contribution. Default null/empty = no subject authority (status quo; producer-only authority). Orthogonal to cohort_scope AND delivery_mode — see CC 2. 3.3.
delivery_mode	no	'pull \
listed	no	Per-membership opt-in flag — value public. Default absent (roster is producer- + self-queryable, NEVER globally enumerable). Distinct from a public community's own community:listing:v1 row (CC 3.2), which advertises the room and names no roster. Public listing mirrors the CC 4.5.9.1 location opt-in discipline: opting into roster visibility is a one-way disclosure the member chooses; substrate does NOT solicit. Composes with CC 4.4.3.2 Policy M community membership and the new CC 5.3.3 streaming endpoint set.
history_on _join	no	'full \

epistemic_mode vs witness_relation — distinct dimensions: these co-vary at edges but

name different concerns. `epistemic_mode` names the *process* by which the claim was formed; `witness_relation` names the *relational position* of the attester to the attested. F-3 detector attestations carry both (`epistemic_mode: derivative` + `witness_relation: derived`). Most encyclical-sourced translations are `witness_relation: external` + `epistemic_mode: hearsay`. When in doubt, set both.

2.1.1 Forward-compatibility rule

The envelope is meant to grow without breaking the peers already speaking it. Two rules make that safe: the canonical-bytes contract (so a new field never silently changes what an old signature covers) and an unknown-field discipline (so a consumer never rejects an envelope merely for carrying a field it has not learned yet).

> ***Canonical-bytes contract**: the canonical-bytes encoding of this envelope for signing follows CC 2.6.1 (JCS over the envelope object; defaults are interpretation-time, NOT encoding-time; relay MUST preserve member presence/absence exactly as the producer signed). Optional fields with documented defaults in the table above ride the CC 2.6.1.1 omit-vs-materialize rule. Conditional-required fields are NOT optional-with-default and substrate rejects mis-shape per CC 2.3.1 + CC 4.5.2.1 + CC 4.5.12.2.*

A Conforming Consumer that receives an envelope carrying a field-name it does not recognize MUST:

- Preserve the unknown field on read (do not strip).
- Preserve it on re-emission if the Consumer is also acting as a Producer relaying the attestation.
- NOT use it in verdict composition.
- NOT reject the envelope on the basis of the unknown field alone.

Producers introducing a new envelope field MUST follow the CC 2.6.4 versioning rules: a new field with a documented default is a MINOR bump; a field whose absence breaks consumer semantics is a MAJOR bump.

2.2 Conformance levels

Interoperability needs named roles so that "conforming" means the same thing to every peer. CEG is the **CIRIS Epistemic Grammar**, the wire grammar this specification defines (CC 8.1.1); the roles below are named for it. A **Producer** is any peer that emits Contributions onto the federation wire. A **Consumer** is any peer that reads and composes verdicts over received Contributions. A **Substrate Implementation** is the storage + transport + crypto layer (CIRISPersist + CIRISEdge + CIRISVerify) underneath both.

Three normative conformance profiles set the floor each role must meet:

1. **CEG-Conforming Producer (CCP)** — emits well-formed envelopes per CC 2.1, signs per CC 2.6.5 References [hybrid-sig], respects reserved-prefix rules per CC 3.4, declares its `oversight_mode` and `witness_relation` per CC 2.1.

2. **CEG-Conforming Consumer (CCC)** — verifies hybrid signatures, enforces reserved-prefix rules at admission, implements at least Policy A (CC 4.4.3.8) with the default aggregation rules from CC 4.4.2, MUST honor `null` placeholder/dev hardware-class rejection per CC 4.2.2.
3. **CEG-Conforming Substrate (CCS)** — implements the storage + transport guarantees referenced in CC 5.3.2 + CC 5.3.1, including idempotent replication, full-SHA blob verification before consumption (CC 5.3.2), and witness-quorum multi-party admission per CC 5.3.1.

Sections that follow MAY add per-feature conformance subsections; the three profiles above are the minimums.

2.3 `subject_key_ids` semantics

Consent is incomplete if only the producer of data has authority over it. The baseline envelope encoded **producer authority** alone (`attesting_key_id`); `subject_key_ids` adds the missing half — **subject authority** — for content where the subject of the data is not the producer of the data. This is the wire-format expression of the Autonomy and Non-maleficence principles: a person who appears in someone else's data can still pull it.

2.3.1 Subject-bearing dimensions (governance requirement)

Per CC 4.5.2, dimensions whose namespace pattern names a subject (e.g., `observed:user:{key_id}:*`, `epistemic:about:{key_id}:*`, `consent:partnered:{user_key}`, `agent_files:*:{subject_target}`) MUST carry `subject_key_ids` containing that subject. The substrate MAY reject admission of subject-naming dimensions that omit `subject_key_ids`. This closes the default-leak failure mode where subject-bearing content publishes without wire-level subject authority.

2.3.2 Federation-key vs canonical-hash identifier

A subject is not always a federation-enrolled identity. The two forms below let `subject_key_ids` name either an enrolled key that can sign for itself or an external party that cannot — without losing the external party's revocation rights.

`subject_key_ids[i]` MAY be either:

1. **A `federation_keys.key_id`** — the subject is a federation-enrolled identity that can sign on its own behalf. Direct revocation: subject signs a `withdraws` against this Contribution; substrate admits via rule (2) of CC 2.4.1 broadened `withdraws` admission. Wire form: the bare `key_id` (no tag).
2. **A tagged canonical-hash identifier** — the subject is an external party with no federation_keys row (Discord user-id, channel-id, content-sha256-bound entity, etc.). The substrate cannot verify a signature from this entity directly; **revocation rides a `delegates_to` proxy chain** per rule (3) of CC 2.4.1 broadened admission. Wire form: a tagged string per CC 2.3.2.1 below.

This answers the open question of how to attest about un-enrolled parties — canonical-hash or pseudonymous federation_keys — with "both, distinguished by an explicit tag on the canonical-hash variant."

2.3.2.1 Canonical-hash wire form + preimage convention

A CEG-Conforming implementation cannot interoperate without pinning two things: (a) how a canonical-hash entry is distinguished from a `federation_keys.key_id` entry on the wire, and (b) what string is hashed (the preimage). Both are normative.

Wire form — tag REQUIRED. A canonical-hash entry MUST carry the tag `canonical:{hashalg}:{hex}`:

- `{hashalg}` — the hash algorithm; `sha256` is the v1 algorithm. Algorithm-agility: future hashes (e.g. `sha3-256`) extend this position without ambiguity
- `{hex}` — the digest in CC 2.6.3 canonical hex (lowercase, unpadded, byte-length-exact: 64 chars for sha256)
- Example: `canonical:sha256:ff7c5632dae6ef3ae7f6283bd35268bc7910332414aa8a1c35a1645ca0295f61`

Why the tag is mandatory and bare-hex is rejected: a `federation_keys.key_id` **was**, in the pre-FSD-003 reference Registry, `hex(sha256(ed25519_pubkey))` — a **lowercase 64-char hex string** — and legacy directories still hold such ids; since FSD-003 a `key_id` is the `<label>-<fingerprint>` form of CC 2.6.8, which contains `-` and base32 letters and is therefore format-distinguishable from a bare hash by construction. The tag stays mandatory because the legacy ids exist and because a subject reference MUST never depend on a consumer's guess about which form it is reading. A bare canonical-hash (also lowercase 64-char hex) would be **format-indistinguishable** from a key_id. A "base64 key_id vs hex canonical-hash" disambiguation heuristic FAILS because Registry key_ids are themselves hex fingerprints, not base64 pubkeys. The tag is therefore load-bearing, not cosmetic. The CC 2.6.3 hex rule governs the `{hex}` segment only; the `canonical:{hashalg}:` prefix is a tagged-union discriminator outside CC 2.6.3's scope and is exempt from the "no separators" rule. **Scope of the fingerprint check:** a consumer verifies a `key_id`'s fingerprint suffix against the public key it obtains **at registration and at first contact** — the proof-of-possession door — for fedIDs and roster keys (CC 3.3.6.3); a held key registered under a pre-FSD-003 or otherwise non-matching id is not re-keyed or refused by a migration (the record is history), it simply cannot be re-registered or introduced at first contact under that id; the bundle-pinned anchors (`ciris-canonical-1`, `humanity-accord*`) and family ids are not fingerprint-suffixed and are verified by the genesis bundle pin (CC 5.3.4), not by this rule; and the fingerprint rule is separate from the roster-id *form* rule of 3.3.6.3 — a `device-<hex>-<fp>` id passes the fingerprint check and is refused by the roster door for its form, on the roster door's own schedule.

Preimage convention — `{platform}:{entity_kind}:{id}`. The string hashed to produce `{hex}` MUST be:

```
preimage = "{platform}:{entity_kind}:{id}"
hex = sha256_hex_lowercase(utf8_bytes(preimage))
```

Parsing is **split-on-first-two-colons**: the substring before the first `:` is `{platform}`; the substring between the first and second `:` is `{entity_kind}`; **everything after the second `:` is `{id}` verbatim, and MAY itself contain colons** (so Matrix IDs like `@alice:example.org` survive). Rules:

- `{platform}` — REQUIRED; MUST match `[a-z0-9][a-z0-9._-]*` — ASCII lowercase, and therefore **colon-free** by construction. Open vocabulary per CC 4.5.1.1. Canonical

seeds: `discord`, `slack`, `twitter`, `matrix`, `email`, `phone`, `github`, `xmpp`, `irc`

- `{entity_kind}` — REQUIRED; the same production as `{platform}`. Canonical seeds: `user`, `channel`, `guild`, `room`, `group`, `address`
- `{id}` — REQUIRED; the platform's **stable, immutable** identifier, **verbatim** (case-preserved — some IDs are case-sensitive). MUST be the immutable account/object identifier (Discord/Twitter numeric snowflake; Matrix MXID; UUID), **NOT** a mutable handle/username/display-name. Using a mutable handle breaks subject-identity stability the moment the user renames. `{id}` is the **only** component that MAY contain a colon.
- **Production (normative)** — `{id}` MUST be a non-empty sequence of Unicode **scalar values**, and MUST NOT contain any surrogate code point (U+D800–U+DFFF), any C0 or C1 control (U+0000–U+001F, U+007F–U+009F — so NUL, TAB, LF and CR are all rejects), or U+FEFF. It MUST NOT begin or end with a character bearing the Unicode `White_Space` property. Interior whitespace is admissible (visible and platform-stable); leading or trailing whitespace is not (invisible, and it would silently fork the digest). No length bound is imposed: the preimage never rides the wire — only its 64-hex digest does.
- **Why scalar values and not "any UTF-8-ish string"** — `utf8(id)` MUST be **total**, and it is undefined for an unpaired surrogate: a Rust implementation cannot construct the string at all, while a Python implementation using `surrogatepass` emits WTF-8 and computes a **different** digest from the same nominal input. Admitting only scalar values makes the encoder agree across implementations by construction. An implementation handed an unpaired surrogate MUST reject it under P1 and MUST NOT substitute U+FFFD — substitution is a repair.

Preimage bytes (normative). The hashed octets are exactly, and only:

```
preimage_bytes = utf8(platform) || 0x3A || utf8(entity_kind) || 0x3A || utf8(id)
hex            = lowercase_hex( SHA-256(preimage_bytes) )
```

`0x3A` is the ASCII colon, contributed exactly twice by the construction. There is **no** length prefix, **no** domain-separation label, **no** trailing NUL, **no** BOM, and no framing of any other kind — an implementation that adds any of these computes a different subject and its gate diverges from every other gate.

P1 — no normalization at hash time. A producer MUST NOT case-fold, trim, pad, or Unicode-normalize (NFC / NFD / NFKC / NFKD) any component before hashing; `{id}` is hashed exactly as the platform issues it. An input that does not already satisfy the productions above MUST be **rejected, not repaired** — a repairing producer and a rejecting producer would otherwise compute different subjects from the same input, which is the divergence this clause exists to close.

P2 — injectivity is structural, not statistical. Because `{platform}` and `{entity_kind}` admit no colon, the joined string reverse-parses to exactly one triple (the `{id}` production only shrinks the admitted domain, so injectivity survives it a fortiori). `("p","k:a","b")` and `("p","k","a:b")` are not two spellings of one preimage: one is an admissible triple, the other is a rejected input. The delimiter therefore carries the full anti-collision property over the admitted domain, and no length prefix is needed to obtain it. The `lp(x) = u32_be(len(x)) ‖ utf8(x)` discipline is the CC 6.1.3 rule for **substrate framing objects that never enter JCS**; a canonical subject is a string value *inside* a CC 2.1 envelope field and sits on the JCS side of that boundary, so adopting `lp` here would cross the CC 6.1.3 seam and re-spell every digest below to buy a property P2 already gives. The delimiter form stands.

P3 — the tag and the digest are governed, not re-opened. `{hex}` is CC 2.6.3 lowercase hex; `{hashalg}` is exactly `sha256` for v1; an exact tag admits neither case variance nor whitespace. Uppercase hex, `SHA256`, a non-v1 algorithm, a bare untagged hex string, and any leading or trailing whitespace are all **rejects**.

Together these make the rule-(3) `delegates_to` proxy chain (`canonical_hash ∈ T.subject_key_ids`) match across producers: every CEG-Conforming producer naming the same `(platform, entity_kind, immutable_id)` triple computes the same `{hex}`, hence the same tagged wire string.

Conformance vectors:

Preimage	sha256 hex	Wire form
`discord:user:1234567 89012345678`	`ff7c5632dae6ef3ae7f6283bd35268 bc7910332414aa8a1c35a1645ca029 5f61`	`canonical:sha256:ff7 c5632...0295f61`
`discord:channel:9876 54321098765432`	`af23411c3c6faa55a788660ea29719 669b9c4e4ea4b6ab9568247d9f646f 05dd`	`canonical:sha256:af2 3411c...646f05dd`
`matrix:user:@alice:e xample.org` (id contains colons)	`16d4d0bf478835a9af68cdaac730a2 9b36f82bf0dfe2073237ee4980f6b9 75d9`	`canonical:sha256:16d 4d0bf...f6b975d9`
`twitter:user:1455079 377986420736` (numeric id, NOT @handle)	`10243ba010bf159a45197d368f91c0 25ef6ac1eb7f42ca32e55b414d90c8 61c2`	`canonical:sha256:102 43ba0...90c861c2`
`email:address:alice@ example.org`	`04481a02fccfc8d99a47bde4f0563d d360d425b0734e8fcd8d5dd7198d0a 263f`	`canonical:sha256:044 81a02...98d0a263f`

Reject vectors — each MUST be refused at the gate, never normalized into acceptance:

Input	Rejected because
`ff7c5632...0295f61` (untagged)	the tag is REQUIRED; bare hex is format-indistinguishable from a `key_id`
`canonical:md5:...`	`{hashalg}` is not the v1 algorithm
`canonical:SHA256:ff7c5632...`	the alg segment is not exactly `sha256`
`canonical:sha256:FF7C5632...`	`{hex}` is not CC 2.6.3 lowercase
`canonical:sha256:ff7c5632...␣` (leading/ trailing space)	an exact tag admits no padding
triple `("Discord","user","1234...")`	`{platform}` is not ASCII-lowercase — reject, do NOT case-fold (P1)
triple `("discord:x","user","1")`	`{platform}` contains `:` (P2)
triple `("discord","user","")`	`{id}` is empty; the production requires ≥ 1 scalar value
triple `("discord","user","1234␣")` (trailing space)	`{id}` ends in `White_Space` — reject, do NOT trim (P1)
triple `("discord","user","12⟨LF⟩34")` (embedded LF; likewise CR, TAB, NUL)	`{id}` contains a C0 control
triple `("matrix","user","@a\uD800:example.org")` (unpaired surrogate)	not a Unicode scalar value — `utf8(id)` is undefined; reject, never WTF-8 and never U+FFFD

2.3.2.2 Rule-(3) proxy vs `canonical_binding`: distinct mechanisms

These are **two distinct mechanisms**, not one:

- **Rule-(3) `delegates_to` proxy** (CC 2.4.1.1 admission rule 3) — *ongoing* revocation authority for an un-enrolled subject. A federation-enrolled key (typically the agent holding data on behalf of the external party) carries `delegates_to(canonical_hash → agent_key, scope: [consent_revocation])`; the agent proxies revocation. The subject never enrolls; the agent acts for them indefinitely.
- `canonical_binding` — *retroactive identity claim.* A now-enrolled federation key asserts "I AM the entity behind `canonical:sha256:{hex}`", binding past canonical-hash subject entries to its real identity. After an admitted `canonical_binding`, the formerly-un-enrolled subject can sign `withdraws` **directly** under rule (2) — the binding promotes the canonical-hash to a real key_id for admission purposes.

`canonical_binding` is **NOT a new admission rule** (no "rule 5"). It composes: the binding is itself a `delegates_to`-shaped attestation (`delegates_to(canonical_hash → newly_enrolled_key, scope: [identity_binding])`) admitted because the enrolling key proves control of the preimage out-of-band. Once bound, rule (2) [direct subject revocation] becomes available to the bound key, and rule (3) [proxy] is no longer needed for that subject.

2.3.2.3 `subject_kind` is a payload-level discriminator

`subject_kind` (e.g. `consent_record`, `consent_replication` (CC 3.3.7), `key_grant`, `takedown_notice`, `community`, `family`, `identity_occurrence`, `location_proof`) is a **payload-level** field — it lives inside the Contribution payload, parallel to how `external_content` carries `sub_kind` in payload. It is NOT an envelope-level field. The envelope-level fields are exactly those in the CC 2.1 table (`cohort_scope`, `subject_key_ids`, `community_id`, `family_id`, `delivery_mode`, etc.); `subject_kind` is the payload discriminator that selects which CC 3.3 payload schema applies. The CC 3.3.5 `consent_record` example and any conformant producer payload agree byte-for-byte: `"subject_kind": "consent_record"` is a payload member.

2.3.2.4 Bilateral ratification is consumer-policy

Per CC 3.3.5 step 4: a bilateral partnership is "ratified iff both halves present under the same `bilateral_pair_id` with `stance: granted`" — and this predicate is **consumer policy**, NOT registry-normative. Ingest-layer builders produce the two halves WITHOUT enforcing ratification. The substrate admits each half independently; composition of the pair into a ratified partnership is a downstream read-time computation, never an admission gate.

2.3.3 Orthogonality with `cohort_scope` AND `delivery_mode` (3-axis)

The envelope keeps visibility, revocability, and delivery as three independent concerns so they can be reasoned about — and enforced — separately. They compose without overlap:

Axis	Field	Authority	Names
Visibility	`cohort_scope` + (`family_id` or `community_id` when set)	Producer-side	Who can SEE the data
Revocability	`subject_key_ids`	Subject-side	Who can REVOKE the data
Delivery	`delivery_mode` + `listed` + `history_on_join`	Substrate / subscriber	Who actively RECEIVES the data + how the substrate fans out

All three may coexist on the same Contribution:

```
A ‘cohort_scope: family‘ contribution
 carrying ‘subject_key_ids: [user_canonical_hash]‘
 carrying ‘delivery_mode: push‘ + ‘history_on_join: from_join‘
 carrying ‘family_id: <acme_household>‘

publishes the bytes at family-cohort visibility (cohort_scope);
the user retains revocation authority (subject_key_ids);
the substrate actively fans out to currently-reachable members
 with new-members getting forward-only content (delivery_mode + history_on_join);
the named family roster gates the membership set (family_id).
```

This orthogonality is load-bearing for the multi-occurrence consent shape: subject-side revocation applies federation-wide regardless of producer's `occurrence_id`; per-occurrence lifecycle consent is a producer-side concern, separately tracked.

The delivery axis is the **third orthogonal axis**: visibility + revocability alone give the substrate no envelope-level handle on who actively *receives* content and how the substrate fans out. Per CC 5.3.3, three optional envelope fields + the CC 5.3.3 endpoint section + a delivery extension to CC 4.4.3.2 Policy M supply it.

2.3.4 Self-as-subject ceremony

When `attesting_key_id` ∈ `subject_key_ids`, the Contribution is a **self-consent ceremony** — the same identity is attesting AS subject AND producer. This composes naturally with the CEG-native agent's self-attestation pattern: agent attests `identity:current` about itself with `subject_key_ids = [self.key_id]`, asserting consent-authority over its own identity claims (D08 autonomy claim).

2.3.5 The shape

`subject_key_ids` is an OPTIONAL list. When present, each entry names a party with substrate-recognized authority over this Contribution's continued processing. The basic shape:

> *A consent record is a signed declaration by a subject about the substrate's continued processing of content where that subject appears.*

The medical record, photo, interview, training-datum, and group-chat cases all share the same shape — a producer publishes a Contribution; one or more subjects appear in it; the subjects retain revocation authority. A single shape carries thirteen distinct content cases uniformly.

2.3.6 Empty/absent = status-quo

`subject_key_ids: null` or `[]` is the status-quo shape (producer-only authority; same as a baseline Contribution carrying no subject authority). Consumers that don't read the field see status-quo behavior. Subject authority is additive at the envelope layer — adding it never changes how an existing consumer reads an envelope that omits it.

2.4 The primitive set: 1+4

The whole grammar reduces to five wire primitives: one workhorse that makes claims, and four structural composers that operate on the claim graph itself. Minimal-and-adequate is the design discipline — fewer primitives mean a smaller surface to attack, audit, and prove conformant, which is what Integrity asks of the wire.

The row-type slot is closed: the five, and nothing else (normative). Every row on the attestation plane names its type in one slot: the row's `attestation_type`, mirrored in the signed envelope. That slot admits exactly the five primitives of this section, closed by CC 1.7. There is no second class of row. A **carrier** — a self-attestation with no score, no subject and no dimension, read by no fold — is a claim with its family rule missing. A holder directory without a family has no declared audience, so bounded content would advertise its existence to everyone, and a fold would have to exclude it by type; the holding claim is `custody:ack:v1` at the content's scope (CC 3.1.3.3). A set of wraps in one row has, as its only structural property, a union that only grows, which one `scores` row per recipient already gives; a key grant is `key_grant:{axis}:v1`, one row per recipient (CC 3.3.2). A row naming the retired carrier token `holds_bytes` as its type is refused `attestation_type_unregistered`.

The gate. A substrate MUST refuse at admission, with `attestation_type_unregistered`, any row whose type is not one of the five, byte-compared — on local emission and on replication alike. The gate is an **allowlist** built from the registry's `_meta.row_types` (CC 3.1.7); the absence of a check on a free-text column is not a gate, and a type that is merely *storable* is not thereby admitted. A row of the five that names a retired carrier token as its `dimension` is refused.

The dimension gate follows the dimension, not the row type. A `dimension` on **any** of the five — the job label on a `delegates_to` row as much as the family on a `scores` row — MUST resolve through the namespace exactly as it does on `scores`: the case rule, the version tail, the reserved and gated stems. A row type is never a way around the namespace.

No more of them. The slot is amended only by a change to this section, in a release of this document, through the CC 1.7 refutation bar: a claim that cannot be expressed by the five plus envelope composition. A component that needs a new row type files it here **before** it ships; a type found in code and not here is a defect in the code. A row of an unregistered type that a substrate already holds when it adopts this rule is reported to the operator, is neither served nor replicated, and is not deleted. The registry carries the closed list and vectors for it, and `tools/audit_emitted_dimensions.py` reports any type constant in a checkout that the list does not admit.

Cosigned rows: quorum is an envelope property, not a row type (normative). A single-signer row cannot carry a quorum's act, and CC 2.4.1.1 rules 1 and 2 would let one signer or one subject undo one; cosigned rows close both, so group records, membership widenings and removals, partner records and the `ciris-canonical` scrub set are rows on the plane. **Any of the five MAY carry `additional_scrubs[]`** (CC 2.1, CC 2.6.1.7): hybrid signatures over the same canonical bytes, beside the row, each by a key whose role the receiver's fold names — the general form of the scrub set the ceremony plane already uses (CC 3.2 T2). The **family declares its `consensus_protocol` source**: the group the row is about, the accord, or — for `licensure:{community_key_id}` (CC 2.4.1.2.1) — the authority community named in the dimension. **Admission:** the row is admitted when its cosigners satisfy that protocol, judged on the *receiver's own* fold at the row's signed instant. **Retraction of a cosigned row is by the same quorum:** a cosigned `withdraws` under the family's protocol; rules 1 and 2 do not apply to it, so no single signer and no subject can retire a community, un-name a charter or retract a roster

event. What this brings onto the base, each as a registered family in CC 3.1.3: **roster versions** (`group:roster:v1`, the T6 head, with `prev_head_digest` and `charter_digest` as envelope members), **membership events** (`membership:widening:v1` / `removal:v1` / `resignation:v1`, cosigned, beside the single-signer proposal, acceptance and decline), and the **key plane** — a key claim (`key:record:v1`, attester = the key, admitted by **proof of possession** against the keys it carries, the one stated exception to resolve-the-attester; rotation is `supersedes`; an anchor scrub or accord co-scrub is a cosignature), occurrences (`identity:occurrence:v1`, signed by the identity or a live occurrence) and revocations (`key:revocation:v1`, never withdrawn, subject = the key digest, `revoked_after` only ever lowering). The roster fold, the acceptance gate and T7 recovery are unchanged; only their inputs become rows. **Admission order:** a row is admitted only once its signer's key-claim row is held, and a row signed by a key not yet held is retryable. What stays off the plane, by definition, is not a claim: payload bytes (sealed chunks, and the A/V plane's symbols), transparency-log heads and cosigns (CC 5.3.1 framing), the wholeness corpus, node-local derived state, an agent's private runtime. **Accord evidence** (proposals and participations as rows) is not on the base: it would amend the entrenched CC 4.2 surface, which requires the founder's explicit ratification, and any such change is bound by this constraint — the invocation's domain-separated canonical bytes MUST travel inside the envelope unchanged, and the kill switch MUST NOT depend on any plane gate beyond verifying those bytes and the holder roster.

2.4.1 The four structural composers

The four structural composers act on the attestation graph itself, not as score-claims on entities. They are how an attester delegates authority, replaces a row, retracts a row, or admits a row was false:

`attestation_type`	**What it does**	**Envelope shape**
`delegates_to`	A authorizes B to sign on A's behalf within a bounded scope	`{scope[], delegation_purpose, delegation_valid_from, delegation_valid_until}` — `delegation_valid_until` is **REQUIRED, at most 365 days (365 × 86 400 s) after `delegation_valid_from`**, on any edge carrying `sub_delegation`, `license`, `grant`, `moderate`, `takedown`, `review` or `slash`, or sitting in an affiliation's `hierarchy` (CC 4.4.3.2.8 C); a perpetual or over-ceiling edge is refused `delegation_term_required`; `delegation_valid_from` absent means the edge's own `asserted_at`, and `delegation_valid_until` absent on an edge that needs no term is open-ended, never malformed — `scope` is the CC 2.1 member (a bare token or a set; the one name for this wire member, never `delegated_scope[]`)
`supersedes`	This attestation row replaces a prior one by the same attester	`{references_attestation_id, supersession_reason, differs_in[]}`
`withdraws`	I retract my prior attestation (does NOT claim it was false)	`{references_attestation_id, withdrawal_reason}`
`recants`	My prior attestation was false at issuance — admits epistemic error	`{references_attestation_id, recantation_reason, what_was_false}`

Translation implications:

- A **doctrinal-development** claim ("this version extends but does not contradict the prior version") is `supersedes` with `differs_in: ["scope", "evidence_refs"]` — NOT `recants` (which would assert prior was false).
- An **acknowledged-error** claim ("the prior framing was wrong; I admit the mistake") is `recants` — distinct from `withdraws` (which retracts without making a falsity claim).
- A **prudent-retraction** ("I'm withdrawing without claiming it was false; context has changed") is `withdraws`.
- An **authority-source claim via delegation** ("this constitutional position derives from authority-key X in scope Y") is `delegates_to` with X as `attested_key_id` (the CC 2.4.1.2 reuse pattern for authority-source claims, replacing what would otherwise need a `grounding: {tradition}:{principle}` prefix that fails CC 1.2 T2).

2.4.1.1 `withdraws` admission rule (semantic, not structural)

Revocation has to reach beyond the original producer without inventing new structure. The `withdraws` admission rule broadens *who* the substrate accepts a retraction from — federation-enrolled subjects, proxies for un-enrolled subjects, and delegates — while keeping the primitive itself unchanged.

Substrate MUST admit a `withdraws` Contribution against target `T` when the issuer's `key_id` satisfies **ANY** of (one carve-out: the anti-Goodhart retraction dual (stated here in full; its consent-and-scoring context is CC 3.4.5 and its contest path is CC 4.5.5) — a subject-authority path, rules 2–4, MUST NOT withdraw a third-party `capacity:*` or `detection:*` row about itself, nor a `licensure:*` or `revocation:*` row about itself (a licensee's contest path is `reconsideration:{grounds}`, never withdrawal); selective erasure of adverse evidence is reputation laundering, and the subject's contest path is CC 4.5.5 `reconsideration:{grounds}`, on which the subject holds standing):

`revoked_after` — the revocation history bound (normative, optional member). A key-plane revocation MAY declare `revoked_after: rfc3339` — a bound before which statements signed by the revoked key **still stand** (the DigiNotar shape: everything after T is suspect; everything before is not unmade). Resolution is a **pure read-time function**: a claim signed by K is valid iff its `asserted_at` precedes `K.revoked_after`, so a partitioned node converges when the revocation arrives late — the same fold shape as consent. Absent member = revoke-all. **`revoked_after` is an upper bound, not a fact** (the DigiNotar intrusion long predated its detection; RFC 5280 separates `invalidityDate` from `revocationDate` for exactly this): a later revocation MAY lower it and MUST NOT raise it, and where a key carries multiple revocations the effective bound is the **minimum** `revoked_after` across all of them, absent-member sorting as negative infinity — keeping the fold commutative and the resolution a pure read-time function, so partitioned convergence is preserved.

A key revocation names its subject by the key's digest, not its `key_id` (normative). A `key_id` is `<label>-<fingerprint>`, and the label is the holder's own keystore alias in cleartext. Revocations replicate federation-wide, so a revocation addressed by `key_id` would tell every node the subject's label — `frank-laptop` — whether or not that node held the key's record. The revocation is a `key:revocation:v1` row (CC 3.1.3), never withdrawn. The subject is therefore **`revoked_key_sha256_ed25519_raw`**: the SHA-256 of the revoked key's raw 32-byte Ed25519 public key, 64 lowercase hex digits (CC 2.6.3) — the full digest whose truncation is the `key_id`'s fingerprint, stable for the life of the key and carrying no label. The member is REQUIRED.

`revoked_key_id` is OPTIONAL: when present it MUST name a key whose raw-key digest equals the subject, a mismatch is refused, and a receiver that does not hold the named key cannot compare the two and refuses the row *retryable*; when absent it is bound as JSON `null`. The signed revocation binds **eight** members — `revocation_id`, `revoked_key_id` (string or `null`), `revoked_key_sha256_ed25519_raw`, `revoking_key_id`, `reason` (string or `null`), `revoked_at`, `effective_at`, `scrub_timestamp`. Three consequences follow. **(1) A revocation may precede the record it revokes**: one naming only the digest MAY be admitted for a key the receiver does not hold, and takes effect the moment a key with that digest is held; authority on this plane is the revoker's and is judged without the subject's record. **(2) Readers ask by digest**: "is key K revoked?" is answered over K's digest, so one revocation reaches every `key_id` that names the same public key, and the per-subject anti-rollback ordering (CC 5.3.2.3) keys on the digest. **(3) Self-revocation** is recognised when the revoker's own key digests to the subject, whether or not `revoked_key_id` is present. A `moderation:*` score MAY likewise name its subject key by `subject_sha256_ed25519_raw` (same spelling; a malformed value is refused) instead of exposing its label. Family and community rosters list members by `key_id`, so a roster's holders see the labels, and roster-removal rows are addressed by `key_id` — they reach only the audience that already holds the roster.

#	Authority path	Description
1	`issuer.key_id == T.attesting_key_id`	Producer self-withdraw
2	`issuer.key_id` ∈ `T.subject_key_ids`	Federation-keys subject revocation
3	∃ `delegates_to` chain: `issuer` →* `canonical_hash` where `canonical_hash` ∈ `T.subject_key_ids` AND `scope` ⊇ `{consent_revocation}`	Proxy authority for non-federation-enrolled subjects
4	`issuer` holds valid `delegates_to` → `any of 1-3`	Delegated revocation (existing primitive, new admission path)

Rule (3) covers the un-enrolled-party case. When a subject is a Discord user-id, a content-sha256-bound entity, or any other non-federation party, revocation authority is mediated through a `delegates_to` chain from a federation-keys signer (typically the agent that holds data on behalf of the external party) to the canonical-hash subject. The agent emits `delegates_to(canonical_hash → agent_key, scope: [consent_revocation])` at proxy-establishment time; subsequent `withdraws` from the agent against any Contribution carrying `canonical_hash` in its `subject_key_ids` is admitted under rule (3). The `canonical_hash` here is the tagged `canonical:{hashalg}:{hex}` wire form with the `{platform}:{entity_kind}:{id}` preimage convention pinned at CC 2.3.2.1. **Rule (3) proxy is distinct from `canonical_binding`** (a retroactive identity claim that promotes a canonical-hash to a real key_id, enabling rule (2) direct revocation) — see CC 2.3.2.2 for the distinction; `canonical_binding` is not a new admission rule.

Rule 0 — a cosigned row (normative). A target `T` that carries `additional_scrubs[]` (CC 2.4) is withdrawn only by a `withdraws` that is itself cosigned under `T`'s family `consensus_protocol`, judged on the receiver's fold at the withdrawal's signed instant; rules 1–4 below do not reach it, so a quorum's act is undone only by a quorum. A `key:revocation:v1` row is withdrawn by nobody (its family rule: revocation is non-rollbackable, CC 3.1.1).

A purge is a tombstone, not a `withdraws` (normative). Rules 0–4 are consent paths — the author's or the subject's. A row the purger neither authored nor is the subject of (a mock-LLM trace, a harness identity's rows, a row no peer can verify) is removed by a **tombstone by digest on its plane**, authorised by a cosigned `takedown_notice` under the trust root's `consensus_protocol` with basis `steward_purge` (CC 3.3.2) naming the selection digest, the rea-

son and the ledger; a substrate MUST NOT admit a `withdraws` forged on the author's behalf for this — a `withdraws` does not even claim falsity, and a purge says the row never belonged. **A purge never tombstones or deletes a key:** a selection naming a `federation_keys` row is refused, a by-actor selection takes the actor's own rows and their derived rows, and key retirement is a separate `key:revocation:v1` signed under this section's own paths — the root's quorum has no path over a key it does not own (a root that could revoke strangers' keys would be an authority roster by another name, CC 3.4.7), so an ownerless key stays registered and inert while its rows go, and the admission gates stop its next ones. Entities go by `revocation:{entity_type}:{reason}`, bytes by the bytes-plane tombstone (CC 4.5.3). **The accountability record is outside the root's reach:** `revocation:*`, `key:revocation:v1`, `moderation:*`, `moderation_track_record:*`, `slashing:*` and `hard_case:*` rows are never purgeable, by the root or by anyone — trust-root moderation is not trust-root history-editing; `hard_case:*` is in the set because an equivocation proof (`hard_case:lineage_equivocation`, CC 6.1.1 N4) or a liveness transition (`community_liveness_stalled` / `_restored`, CC 3.2 T7) is an adverse row about a root whose only purge door is that root's own quorum — the subject — and the anti-Goodhart dual admits no subject-side removal of an adverse row; the substrate that produced the proof withdraws it under rule 1, or recants a false one, and nobody else; a structural composer (`withdraws`, `recants`, `supersedes`) is purged only together with its target, and a `delegates_to` only when a party to it is in the selection — a purge removes a party, never a third party's standing. **Derived rows follow the reason, and the selection digest says which:** a `scores` row whose evidence was never admissible input (mock, harness, unverifiable) is purged with its evidence, in the same selection and ledger — its subject never ran, so it attests nothing — and so are a trace's own `trace:complete:v1` attestation and the bytes a purged holding claim holds, transitively over `evidence_refs`; a score over real evidence later taken down on a legal basis stands, its `evidence_refs` resolving to the tombstone. The door refuses a purge without the notice (`purge_unauthorised`) and one whose selection digest differs from the authorised one (`purge_selection_digest_mismatch`); every purged id is appended to a ledger attested at the next tag.

Per-rule audit metadata: substrate SHOULD record which admission rule (0-4) admitted each `withdraws` Contribution in `federation_attestations` metadata so downstream consumers can compose policy (e.g., higher confidence weight for subject self-revocation rule 2 than for proxy rule 3).

Composition with `recants`: subject-side authority does NOT extend to `recants` (the falsity-admission primitive) — only the original attester can `recant` their own claim. A subject who believes the producer's claim about them is FACTUALLY wrong (not merely unwanted) issues `scores` with negative polarity on a contradicting dimension; that is the consumer-side rebuttal path, distinct from consent revocation. The `recants` / `withdraws` distinction matters precisely because subject authority covers the consent dimension (revocability) but not the truth dimension (falsity).

2.4.1.2 Authority-source claims via `delegates_to`

A constitutional or framework claim can name its source-of-authority by emitting `delegates_to` against an `attested_key_id` representing the framework, with `scope` naming the principle. Example: a Ubuntu-substrate commitment in CC 1.13.1 commitment 2 can be expressed as `delegates_to` against the `ubuntu_relational_substrate` framework-key with `scope: ["personhood_const`
Reuses the existing structural primitive rather than introducing a `grounding:{tradition}:{principle}` prefix (which would fail CC 1.2 T2 — "tradition" claims are interpretive, not mechanism-

descriptive).

Owner-binding is the single-valued sub-relation. A `delegates_to(user → node/agent)` carrying `delegation_purpose: owner_binding` is the **ownership** grant that defines the `self` cohort (CC 3.2 single-owner invariant): at most one is live per owned key at a time, and the substrate rejects a second, distinct-owner binding **at bind time**. This is the **one** `delegation_purpose` that is single-valued — act-on-behalf, hierarchy (CC 4.5.13), and authority-source delegations remain **multi-parent** DAGs. `delegation_purpose` thereby distinguishes the ownership relation from the general delegation grammar with **no new primitive**: the `self`-cohort boundary rides an existing field, not a fifth relation.

2.4.1.2.1 Licensure, grant, delegation: three concepts, three wire discriminators (normative)

`delegates_to` is one of three ways a key comes to hold something, and the three are routinely conflated because only one of them has a structural primitive. Two of them run through different machinery — *"license authority rides the existing CC 3.2 founder-quorum machinery; role authority rides the existing CC 4.4.3.4.3.1 delegation resolver"* (CC 3.3.9); the triad is stated here, at the grammar.

	Licensure	**Grant**	**Delegation**
who gives it	an **authority**	the asset's **owner / steward**	a **principal**
what it confers	standing to practise	access to a specific thing	agency to act *for* someone
what it is about	the subject	the asset (hash, stream, balance)	the principal's own powers
ends by	the authority revokes / suspends	rotation, expiry, exhaustion	the principal withdraws; attenuation
chains?	**no**	**no**	**yes** — attenuated, depth-capped, revocable per link
wire discriminator	dimension prefix on `scores`: `licensure:{authority_id}` / `attestation:license_validity`	`subject_kind` (`key_grant`) or the `consent:scope:*` family	**structural primitive**: `attestation_type: delegates_to`

The transitivity row is the discriminator of record. Only delegation composes, which is why only `delegates_to` gets a graph walk (CC 4.1.1 depth cap, cycle rejection, aggregate-weight cap). A licence does not chain — holding one confers no power to issue one. A grant does not chain — receiving access confers no power to re-grant it (below). Any design that walks a licence or a grant transitively has mistaken it for a delegation.

The professional analogue is exact, and it is the domain this federation licenses: a physician assistant holds a **licence** from a board, practises under a supervising physician's **delegated** authority, and holds **privileges** — a grant — at one hospital for named procedures. Three grantors, three revocation paths, reported as separate categories by the registries that track them.

Provenance is the authority, not a delegation (normative). A licence's provenance is its `authority_id` — carried inside the dimension — and that authority is conferred by **quorum** (CC 3.3.9 / CC 3.2 founder-quorum), **never** by `delegates_to`. Three things must not be collapsed into one another: *who grants the licence* (the `authority_id`), *who vouches for the attester* (the trust walk, `infra:attest`), and *what external legal force it carries* (a **bridge** on an in-grammar record — CC 8.3: bridges compose with the record, they are never alternatives to it).

Who may be a licensing authority, and what admits a licence (normative). Anyone. `authority_id` names a key — a `federation_keys.key_id`, or an organisation whose keys resolve through `org_membership` (CC 3.3.9) — and "X holds licence authority for `A`" means exactly that X's key *is* `A`, or holds a `license`-scoped delegation whose chain resolves to `A` (CC 4.4.3.4.3). There is no roster of permitted authorities and a substrate checks none: a `licensure:{A}` row admits under the same gate as every other row on the wire — the reader trusts the emitter's root or has consented to receive from it (CC 4.4.3.8; the contextual-integrity exemplar: a medical licensure claim reaches a reader only along a flow that reader's trust or consent admits) — and it reaches nobody else. Per CC 4.4.4 a Sovereign key scoring `licensure:CA_medical_board` is wire-identical to a Registry steward scoring it; consumer policy weights by attester. "By quorum" in the provenance paragraph above is a statement about the authority's *own* governance where it is a collective — the CC 3.2 founder quorum that constitutes a community's key, the CC 3.3.9 steward quorum that admits a `partner_record` — never an admission gate a substrate applies to somebody else's authority. **What the fold keys on (normative).** The `licensure:{A}` status set for a subject folds only rows whose emitter resolves to `A` — `A`'s key, or a `license` chain to `A`. A row under `licensure:{A}` signed by a key that is neither `A` nor `A`'s delegate is admitted if trusted or consented to, but it is *testimony about* `A`'s licensure, not `A`'s licensure: it stays out of the `(subject, A)` fold and composes at consumer confidence — which is how a stranger's absorbing `revoked` binds nobody. The one refusal is on the `license` scope itself: an issuance whose delegator chain does not resolve to `A` is `licensure_delegator_not_authority`. There is no bootstrap: `A`'s first `licensure:{A}` row is signed by `A`, or, for a keyless affiliation, cosigned by its quorum. **Authority is judged at issuance:** at the row's signed `asserted_at` under the cosigned-row rule, never at receipt, because this is a shared verdict every node must compute identically; a later `withdraws` of a link in the chain stops future issuance and does **not** reclassify licences already issued — the authority ends those with `suspended` or `revoked`. **A licence has no public default:** it is a claim about a person, so its `cohort_scope` is the authority's declared choice *per licence class* in the affiliation's charter, the smallest that fits (CC 1.13.3.4) — a practice licence meant to be checked by strangers is federation-scope by its nature; an internal credential is not. At whatever scope the class declares, the licence and the **authority head** a verifier needs (the roster version, the protocol, the `license` edges on the chain) travel together; where the class is not public they are **re-emitted** to the readers who must validate — new rows, signed and cosigned again, with the readers in `subject_key_ids` — never carried at the wrong scope (`cohort_scope` is in the signed bytes) and never by widening the roster. An organisation is an **affiliation** (CC 4.4.3.2.8), and an organisation-named authority is its `community_key_id`: its authority set is whatever the affiliation's *current* `consensus_protocol` admits, and a `licensure:{community_key_id}` row is a **cosigned row** (CC 2.4) whose protocol source is the authority named in the dimension — the one case where the source is neither the group the row is about nor the accord. A `license`-scoped chain roots at that quorum; one founder suffices only where the protocol says so.

A grant is non-transferable by default (normative). A `key_grant` conveys access to an asset; it does **not** convey the power to convey it. An onward grant is a **new grant issued by the asset's owner or steward, or by a holder of a `grant`-scoped delegation from them** (CC 4.4.3.4.3), never a forwarded one, and `rotation_chain` is supersession lineage — not onward transfer. **Possession of the data-encryption key is not issuance authority.** Every recipient necessarily holds the DEK, having unwrapped `wrapped_dek`, so the ability to wrap a new grant is *necessary* and never *sufficient*: the cryptographic gate stops a stranger, and only the authority gate stops a recipient. A substrate MUST refuse at admission a `key_grant` whose issuer neither owns or stewards the asset nor holds a `grant`-scoped delegation over it, exactly as it refuses an issuance whose delegator lacks the underlying authority. A design that treats receipt

of a grant — or possession of its key — as authority to re-grant has, again, mistaken a grant for a delegation.

2.4.1.3 The `recants` distinction matters

No prior identity system (PGP, SPKI/SDSI, W3C VC) types epistemic-error-admission as a wire primitive distinct from retraction. CEG types both because the Recursive Golden Rule applies to attesters: admitting error is a primary act, not a derivative of retraction. Consumer policy can apply different trust adjustments to attesters who `recant` versus those who `withdraw`.

2.4.2 The workhorse: `scores`

The federation has exactly **one** workhorse attestation primitive. Every claim about an entity — positive or negative, identity or capability or behavior or state or commitment, by any attester source — is expressed as a `scores` attestation on a named dimension.

```
// Wire shape (Persist's federation_attestations row):
attestation_type: "scores"
attesting_key_id: <attester's key_id>
attested_key_id: <subject's key_id>
attestation_envelope: {
 "dimension": "<canonical-namespace-prefix>:<scoped-leaf>",
 "score": <f64 in [-1.0, +1.0]>,
 "confidence": <f64 in [0.0, 1.0]>,
 "context": "<free-form scoping detail>",
 "evidence_refs": ["<URI or hash referencing backing evidence>",...],
 "valid_until": "<ISO8601 datetime, optional>",
 "epistemic_mode": "<direct | crypto | hearsay | derivative | appeal>", // optional; default 'direct'
 "witness_relation": "<self | external | derived>", // optional; default 'external'
 "oversight_mode": "<HITL | HOTL | HOOTL | null>", // optional; default null
 "occurrence_id": "<occurrence-N | __shared__ | null>", // optional; default null
 "occurrence_count": <int >= 1 | null>, // optional; default null
 "occurrence_role": "<primary | shared | replica | null>", // optional; default null
 "stake": "<free | reputational | capital | cryptoeconomic>" // optional; default 'reputational'
}
```

Full field semantics in CC 2.1.

2.5 The reasoning grammar: the eight axes

The wire stays minimal precisely because the richness lives in how consumers *reason* about it. These eight axes are **not wire fields**; they are the canonical questions a consumer can ask about any Attestation. Nothing in the wire prescribes the verdict — the axes are the vocabulary the verdict is built from.

Axis	Question	Values
Polarity	Direction of the claim	Positive / Negative / Neutral / Indeterminate{reason}
Object	What the claim is about	key_id (entity) / attestation_id / contribution_id
Time	When is the claim valid	`asserted_at` + optional `valid_until`; consumer composes with substrate `expires_at`
Epistemic mode	How was the claim formed	direct / crypto / hearsay / derivative / appeal

Axis	Question	Values
Reversibility	Can the attestation be reversed	rollbackable / non-rollbackable (consumer policy + per-prefix rule)
Stake	What's backing the attester's claim	free / reputational / capital / cryptoeconomic
Scope	At what scale does the claim apply	self / family / community / affiliations / species / biosphere / federation
Inter-attestation relations	How does this attestation relate to others	standalone / refers-to / supersedes / withdraws / recants / corroborates / contradicts / clarifies

`biosphere` is a distinct Scope value from `species` (which is the Homo sapiens cohort). See CC 4.1 for the `planet` colloquial alias note.

2.6 Foreword

CEG — the CIRIS Epistemic Grammar — is the federation's language for making **structured, signed, machine-checkable claims about reality and each other**. It is the wire format the federation's peers speak. Everything above is that grammar; everything below is the canonicalization discipline that makes one peer's bytes verify identically on another peer's machine — the mechanical foundation of the Integrity principle.

The grammar has exactly five wire-format primitives (one workhorse + four structural composers) and an open-vocabulary dimension namespace organized by mechanism-descriptive prefixes. Consumers compose verdicts from primitive attestations using the policies in CC 4.4; nothing in the wire format prescribes what verdict to reach.

CEG is **substrate-consuming**: it sits above the federation substrate (CIRISPersist for storage, CIRISVerify for crypto, CIRISEdge for transport) and below the application tier (CIRISAgent). It does not author primitives in the substrate it consumes; it composes policy over them. It is also **substrate-supplying** for the second-tier consensus crates (CIRISNodeCore for consensus, CIRISLensCore for detection) — they own slices of the dimension namespace and emit attestations that other CEG consumers read.

This specification has **two readerships**:

- **Implementers** of federation primitives consuming or emitting CEG attestations: read CC 1.13-CC 4.5 normative.
- **Translators** mapping substantive content into CEG envelopes: read CC 8.2-CC 8.1 + the `LANGUAGE_PRIMER.md` companion.

Both readerships should read CC 1.13 first.

2.6.1 Envelope canonicalization: JCS + the omit-vs-materialize rule

This rule closes the canonical-bytes ambiguity that arises whenever an envelope carries optional fields with documented defaults — `epistemic_mode`/`witness_relation`/`occurrence_*`/`stake`, `oversight_mode`, `subject_key_ids`, `family_id`, `community_id`. Without it, two honest peers can disagree on the bytes and a valid signature fails to verify.

The hazard is structural. If Producer A omits an optional field (relying on the CC 2.1 documented default) and Relay R re-serializes the attestation with the default materialized into the byte stream, the canonical bytes diverge and the Ed25519 + ML-DSA-65 signatures no longer verify. The rule below makes the discipline explicit and normative: every party — producer, substrate, relay, consumer — must make the same choice, and that choice is "preserve what the producer signed, exactly."

2.6.1.1 The round-trip rule: defaults are interpretation-time, not encoding-time (normative)

The canonical bytes are computed over **the literal envelope members the producer signs**. Optional fields that the producer omits MUST NOT be materialized into canonical bytes by any party — producer, substrate, relay, or consumer. Optional fields that the producer explicitly emits (even at their default value) MUST be preserved in the canonical bytes by any party. Documented defaults from CC 2.1 are **interpretation-time semantics**, not encoding-time content.

Two valid encodings of the semantically-identical attestation:

```
# Producer A -- omits epistemic_mode (relies on S4 default)
{"attested_key_id":"...","attesting_key_id":"...","confidence":0.9,"dimension":"licensure:CA_medical_board
    ","score":0.8}

# Producer B -- explicitly emits epistemic_mode at its default value
{"attested_key_id":"...","attesting_key_id":"...","confidence":0.9,"dimension":"licensure:CA_medical_board
    ","epistemic_mode":"direct","score":0.8}
```

Both producers compute different canonical bytes (Producer B's includes the `"epistemic_mode":"direct"` member) and emit different signatures. **Both signatures verify under their respective canonical bytes.** Both are **semantically equivalent** — a CEG-Conforming Consumer evaluates `effective_epistemic_mode(envelope) = envelope.epistemic_mode if present else "direct"` and proceeds identically. The wire-format admits both encodings; the consumer policy admits no observable difference.

Relay discipline (normative). A substrate, relay, or consumer that re-stores or forwards an attestation MUST preserve member presence/absence exactly as the producer signed it:

- Stripping an explicitly-emitted default ("the producer wrote `epistemic_mode:direct` but I'll save bytes by removing it") MUST NOT happen
- Materializing an omitted default ("the producer omitted `epistemic_mode` so I'll fill in `direct` for clarity") MUST NOT happen
- Reordering object members on re-emission is REQUIRED (JCS lexicographic order; if a producer somehow emits non-canonical order, the relay re-canonicalizes — but member presence/absence stays fixed)

This composes with the CC 2.1.1 forward-compatibility rule (which already mandates preserving unknown fields on read and re-emission); the same preservation discipline extends to known-optional fields.

2.6.1.1.1 Array ordering + byte-field + timestamp encoding (normative: the three determinism rules JCS does NOT pin)

JCS (CC 2.6.1.3) canonicalizes **object member order** but is silent on three things that still let two conformant producers emit different bytes (and therefore non-verifying signatures, the same hazard). All three are pinned here once, globally, for every CEG envelope:

1. **Array element order.** An array field is one of two kinds, stated in its CC 2.1/CC 3.1 definition:

- **Set-semantics** (order carries no meaning) — elements MUST be **lexicographically sorted by their JCS string form (UTF-16 code-unit order, ascending)** before signing. This is producer-independent: any producer building the set in any order yields identical bytes. **`subject_key_ids[]` and `delegates_to.scope[]` are set-semantics → sorted.**
- **Sequence-semantics** (order is meaningful) — elements retain their as-authored order. **`transport_destination.aspects[]`** (RNS hash preimage order), **`key_grant.rotation_chain`** (supersession lineage), and **`evidence_refs[]`** (producer-asserted order) are sequence-semantics → preserved. A field's definition MUST declare which kind it is; absent a declaration, set-semantics (sorted) is the default. A substrate that finds a set-kind array out of order refuses the row with **`federation_set_semantics_unsorted`**, naming the member, and never re-sorts it (re-sorting would move signed bytes). **A set-kind array is sorted AND unique:** a duplicate element is refused under the same token. No reader counts multiplicity anywhere in the substrate, every other set this document encodes is already stated as deduplicated (CC 3.4.7.1, CC 3.3.7), and a duplicate is otherwise a second byte string for one meaning. **An optional member is unset by absence, never by `null`:** an explicit JSON `null` on any optional envelope member is refused as malformed — a null and an absence would be two byte strings for one meaning, the duplicate rule again — refused as `federation_malformed_row` with reason `envelope_member_null` (one of the closed malformed-row reasons of CC 2.6.1.7, beside `envelope_member_missing` / `envelope_member_invalid`). Where this document writes a default as `null` it means *absent*.

1. **Byte-field string encoding.** JSON has no byte type, so every byte/binary field is a string whose encoding MUST be pinned. **Hashes and raw key material — public keys, `destination_hash`, `root_hash`, digests and fingerprints — MUST be lowercase hex per CC 2.6.3** (no `0x`, no separators, byte-length-exact; never base64 in canonical bytes). Fields explicitly documented as base64 (e.g., `hardware_attestation`, an opaque attestation blob) use base64 as their definition states — but the *default* for any key/hash/id byte field is CC 2.6.3 lowercase hex.

- **`subject_key_ids[]` is the one two-form field, and CC 2.3.2.1 governs its second form.** An element naming a `federation_keys.key_id` — like `attesting_key_id` and `attested_key_id` — carries that key's `key_id` string **verbatim**: the CC 2.6.8 `<label>-<fingerprint>` form (lowercase `[a-z0-9-]`), or a legacy 64-hex id a directory still holds; a `key_id` is an identifier, not a byte field, so the hex rule above does not re-encode it. An element naming a canonical-hash subject is the **tagged** `canonical:{hashalg}:{hex}` string, which CC 2.3.2.1 REQUIRES and whose untagged bare-hex spelling CC 2.3.2.1 refuses at the gate. The CC 2.6.3 hex rule governs the `{hex}` segment of that string only; the `canonical:{hashalg}:` prefix is a tagged-union discriminator, outside CC 2.6.3's scope and **exempt from the no-separators rule**. The exemption is restated here on purpose: these are signing bytes, so an implementer who reads only this clause and strips the tag

— or refuses the tagged form — produces a different canonical envelope and a **silently non-verifying signature**, not a visible admission error.

- **The base64 variant is pinned: every field documented as base64 MUST use the RFC 4648 §4 STANDARD alphabet, WITH padding, no whitespace or embedded newlines**. The pin lives at the **protocol layer**: the crypto layer is encoding-agnostic and deals in raw bytes; the wire encoding is CEG's to pin. A variant mismatch (URL-safe alphabet, stripped padding) produces different signed bytes and a **silently non-verifying signature** — the same hazard class as the wrap-algorithm wire-string. The vector set MUST include a base64-variant case (encode→sign→verify round-trip across two independent encoders).

1. **Timestamp encoding.** Every datetime field (`signed_at`, `asserted_at`, `valid_until`, `delegation_valid_from/_until`, `valid_at`, ...) is the **CC 2.6.2 canonical string** — UTC literal `Z` (never `+00:00`), millisecond precision (exactly three fractional digits), `YYYY-MM-DDTHH:MM:SS.sssZ`. A producer emitting `+00:00` or a different sub-second precision produces different bytes and a non-verifying signature.

With key order (JCS) + omit-vs-materialize + these three, **byte-identity across conformant implementations is closed by construction**. The cross-impl JCS vector set MUST cover all three (a set-vs-sequence array case, a hex-vs-base64 byte case, a timestamp case) alongside the per-Contribution vectors.

2.6.1.2 Per-field encoding table (informational)

The omit-vs-materialize rule applies uniformly to every optional CC 2.1 field. Catalog:

Field	Section	Default per CC 2.1	Canonical when omitted	Canonical when explicit
`epistemic_mode`	CC 2.1	`direct`	member absent	`"epistemic_mode":"direct"` (or other enum value)
`witness_relation`	CC 2.1	`external`	member absent	`"witness_relation":"self"` (or other enum value)
`oversight_mode`	CC 2.1	`null` (per-cell default applies)	member absent	`"oversight_mode":"HITL"` (or other enum value)
`occurrence_id`	CC 2.1	`null` → `"occurrence-0"` at interpretation	member absent	`"occurrence_id":"occurrence-1"`
`occurrence_count`	CC 2.1	`null` → 1 at interpretation	member absent	`"occurrence_count":3`
`occurrence_role`	CC 2.1	`null` → `"primary"` at interpretation	member absent	`"occurrence_role":"shared"`
`stake`	CC 2.1	`reputational`	member absent	`"stake":"capital"` (or other enum value)
`context`	CC 2.1	absent	member absent	`"context":"..."` (free-form)
`evidence_refs`	CC 2.1	absent	member absent	`"evidence_refs":["..."]`
`valid_until`	CC 2.1	absent	member absent	`"valid_until":"2026-12-31T00:00:00.000Z"`
`subject_key_ids`	CC 2.3	`null`/empty	member absent	`"subject_key_ids":["..."]`

Field	Section	Default per CC 2.1	Canonical when omitted	Canonical when explicit
`scope`	CC 2.1, CC 3.3.1	absent	member absent	`"scope":["retain","share"]` or `"scope":"retain:90d"`
`session_id`	CC 3.1.3.1	n/a (REQUIRED on `session:*`)	member absent (not a claim)	`"session_id":"s1"`
`claimed_at`	CC 3.1.3.1	n/a (REQUIRED on `session:*`)	member absent (not a claim)	`"claimed_at":"2026-08-30T12:00:00.000Z"`
`custody_state`	CC 3.1.3.3	n/a (REQUIRED on a custody report)	member absent (not a custody report; on one, admission refuses)	`"custody_state":"none"`
`group_kind`	CC 3.1.3.2	n/a (REQUIRED on a membership proposal)	member absent (not a proposal; on one, admission refuses)	`"group_kind":"family"`
`role`	CC 3.1.3.2	`null` (no role offered)	member absent	`"role":"member"`
`expires_at`	CC 3.1.3.2	n/a (REQUIRED on a membership proposal)	member absent (not a proposal; on one, admission refuses)	`"expires_at":"2026-10-07T00:00:00.000Z"`
`proposal_hash`	CC 3.1.3.2	n/a (REQUIRED on a membership answer)	member absent (not an answer; on one, admission refuses)	`"proposal_hash":"<original_content_hash>"`
`attached_head_digest`	CC 3.2 T4a	n/a (REQUIRED on `trust:accepts:v1`)	member absent (not an acceptance edge; a NEW acceptance edge without it is refused; an edge admitted before the member was required stays valid, T4)	`"attached_head_digest":"<64-char-lowercase>"`
`subject_sha256_ed25519_raw`	CC 2.4.1.1	absent (the subject is named some other way, or not at all)	member absent	`"subject_sha256_ed25519_raw":"<64 lowercase hex>"`
`attach_window_secs`	CC 3.2 T4a	absent = no window declared (attach by out-of-band anchor only)	member absent	`"attach_window_secs":604800`
`witness_cadence_secs`	CC 3.2 T6	n/a (REQUIRED on a witnessed lineage's charter)	member absent (the lineage is unwitnessed for T6)	`"witness_cadence_secs":86400`
`additional_scrubs`	CC 2.4	absent (single-signer row)	member absent	`"additional_scrubs":[{"scrub_key_id":"..."," scrub_signature_classical":"...","scrub_signature_pqc":"..."}]` (beside the row)
`halt_fuse_secs`	CC 4.2.1.1	**86400** — a constitutional constant, not a substrate default	member absent ≡ 86400 (a verifier reads the constant)	`"halt_fuse_secs":3600` (an explicit value overrides)
`recovery_commitments`	CC 4.2.6	n/a (REQUIRED on the accord's charter)	member absent (no holder can self-rotate; recovery is a roster change)	`"recovery_commitments":{"<holder_key_id>":"<lowercase hex>"}`
`witness_quorum`	CC 3.2 T6	absent ≡ 0: witnessed mode off; no substrate default is substituted	member absent (witnessed mode off — the lineage is unwitnessed for T6)	`"witness_quorum":2`
`witnesses`	CC 3.2 T6	n/a (REQUIRED on a witnessed lineage's charter)	member absent (the lineage is unwitnessed for T6)	`"witnesses":[<SignedKeyRecord>,...]`
`family_id`	CC 3.3.4	n/a (REQUIRED iff `cohort_scope == family`)	member absent (admission rejects if cohort_scope == family)	`"family_id":"..."`

Field	Section	Default per CC 2.1	Canonical when omitted	Canonical when explicit
`community_id`	CC 3.2	n/a (REQUIRED iff `cohort_scope` ∈ {`community`, `affiliations`})	member absent (admission rejects if cohort_scope == community)	`"community_id":"..."`
`delivery_mode`	CC 5.3.3	`pull`	member absent	`"delivery_mode":"push"`
`listed`	CC 5.3.3	absent (private roster)	member absent	`"listed":"public"` (opt-in only; producers MUST omit unless the subject has opted in)
`history_on_join`	CC 5.3.3	`from_join`	member absent	`"history_on_join":"full"`

The conditional-required fields `family_id` and `community_id` are NOT optional-with-default — substrate rejects mis-shape per CC 2.3.1 + CC 4.5.2.1 + CC 4.5.12.2 — but they encode under the same JCS rule when present.

2.6.1.3 Canonical encoding format (normative)

CEG envelope signing bytes are computed via **JCS — JSON Canonicalization Scheme, RFC 8785** (Rundgren, Jordan, Erdtman; March 2020). JCS pins, in summary:

- Object members sorted lexicographically by member name (UTF-16 code-unit order, RFC 8785 §3.2.3)
- No insignificant whitespace
- UTF-8 byte encoding (RFC 8259 §8.1)
- Strings escaped per RFC 8259 §7 with the JCS narrowing on `\uXXXX` form
- Numbers serialized per the ES6-derived rule (RFC 8785 §3.2.2.3) — integers without trailing `.0`, no exponent unless the magnitude requires it

A CEG-Conforming Producer MUST produce signing bytes via JCS over the envelope object — `attestation_envelope`, whose member set and placement are pinned at CC 2.6.1.7. A CEG-Conforming Consumer (CCC) MUST recompute signing bytes via the same JCS rule for signature verification. A CEG-Conforming Substrate (CCS) MUST preserve the as-received envelope object bytes for relay (per the round-trip rule above); it MAY store a parsed representation alongside, but the canonical-bytes contract is against the as-received **object** — its member set and its member values, which no party may alter — and not against a parsed-and-re-serialized shape with defaults materialized or members dropped. Insignificant whitespace and member order are **not** part of that contract: JCS erases both, so normalizing them changes no signing byte.

Redaction vs tampering — the signed-discriminator shape is excluded (normative dead end). The excluded shape is a signed "I redacted this" marker distinguishing authorized redaction from tampering, because **a signed redaction marker proves an authorized party changed the bytes — it does not prove that *only* the redacted parts changed. **A compromised or coerced authority key becomes a licence to rewrite arbitrary content while every reader reports the row as legitimately redacted: integrity replaced by authority, the AV-50 conflation arriving quietly. Any future redaction scheme is bound by four constraints:** (a) it lives on the envelope plane — **the integrity gate is `original_c`**

`ontent_hash` at ingest, receiver-side; `persist_row_hash` is never re-verified on read, so a storage-plane build is inert; (b) it carries its own redactability declaration **in the signed object — per-field commitments cannot derive "which fields are redactable" from the manifest, whose wire fields and storage containers are disjoint sets with no bridge** — unless the scheme is uniformly redactable by construction **(BBS+-style multi-message signatures, JMSW set schemes), in which case the signed object MUST instead commit to the member count and index ordering;** (c) it lives above whole-row hashing**, which forecloses partial redaction downstream;** (d) the third option stays open** — a redacted row MAY simply be a *different object carrying a different kind*, relocating the work entirely. Note for any per-field commitment design: canonical member ordering is **one of two routes** to a derivable leaf sequence — the other is committing to the encoded disclosure bytes directly (the SD-JWT/mDoc salted-digest shape, which needs no canonicalization); where the canonical route is taken, CC 2.6's canonicalization choice is already load-bearing for redactable signatures.

Number totality (normative). RFC 8785 §3.2.2.3 fixes the number model as the IEEE-754 binary64 (ECMAScript `Number`) image. A CCP MUST coerce every numeric value to its finite IEEE-754 double image *before* computing JCS bytes, and MUST reject — with a hard error, never a fallback or best-effort hash — any number that has no finite double (overflow to ±Infinity, e.g. `1e1000` or a thousand-digit integer). An implementation built against an arbitrary-precision JSON parser otherwise admits a string-backed number that bypasses the double model: JCS then emits a non-canonical result and the content address silently diverges between peers — the same non-determinism-is-broken hazard class as the byte-field and timestamp rules of CC 2.6.1.1.1. With this rule a content address is always either the spec-canonical hash or an honest error, never a wrong-but-plausible one, in any parser feature configuration. The cross-impl JCS vector set MUST include a non-representable-number case (a value with no finite double, asserted to reject) under both default and arbitrary-precision parser builds.

Canonical-bytes size bound (normative). A CEG envelope's canonical (JCS) bytes MUST NOT exceed **1 MiB (1 048 576 bytes)** — the signed thing is the sized thing, so the bound is on the bytes the signature covers, never on a stored or transport-framed image of them. A CCP MUST NOT emit an envelope above the cap; a CCS MUST reject one at admission — at **every** write path, including capsule/FFI and tier-ingest, not only an HTTP body gate — with `ENVELOPE_TOO_LARGE` (HTTP 413, CC 5.3.6.1). The value is the blob-payload threshold (inline below 1 MiB, a blob above), so the cap adds a rule, not a second regime. Above the cap **the envelope carries the manifest, not the bytes**: the heavy payload moves to the blob plane as encrypted chunks, replicated per item to the holder count its owner chose (CC 6.1.5), and the envelope names it by content hash in `evidence_refs`. How that hash resolves is **scope-dependent**, and both routes exist: for Commons scopes it resolves through the holding-claim directory (`custody:ack:v1 here` rows at federation scope, CC 3.1.3.3) + CC 5.3.2 `ContentFetch`; for a bounded cohort the same rows exist **inside the cohort only** — CC 5.2 forbids any holding claim leaving a bounded cohort, unconditionally, and that is the structural-invisibility promise, not a tunable — so a non-member has no directory route at all. There the same `evidence_refs` content hash resolves over the CC 5.2 in-cohort at-rest-encrypted delivery path to admitted self-collective / family members: no directory attestation is emitted and the reference never propagates beyond the cohort. An above-cap self- or family-scoped envelope therefore has exactly one route, not zero. The attestation remains the CEG object; only its payload relocates — **no new envelope field and no new primitive; the CC 1.7 1+4 surface is untouched.** This is a different axis from the CC 5.4.3 constant frame size on a shaped link: that is a traffic-analysis uniformity rule which *fragments* an oversize payload rather than refusing it, so it bounds no admission decision — absent the cap here, a multi-megabyte attestation is simply hundreds more fragments on the anti-entropy walk. The cap must be wire-contract precisely because peers have

to agree on it: unratified, one node's legitimate envelope is another's rejection. The freeze-gate vector set MUST include an at-cap envelope (admits) and a cap-plus-one-byte envelope (rejects with the token above).

What is measured, and when (normative). JCS bytes are not the received octets — the same envelope can arrive pretty-printed, with expanded `\uXXXX` escapes, or with exponent-form numbers — so the cap alone does not tell an implementer when to stop reading, and a rule measured on canonical bytes must not be readable as a mandate to buffer an unbounded input in order to find them. Two bounds, distinct roles:

1. **The contract bound (interop-visible).** `|JCS(envelope)|` $\leq$ `1 MiB`. This is the only bound peers must agree on; it is measured on the bytes the signature covers and is therefore decided **after** parse. A CCP MUST NOT emit above it; a CCS MUST reject above it with `ENVELOPE_TOO_LARGE`.
2. **The intake bound (local, mandatory, not interop-visible).** Because (1) cannot be decided without parsing, every write path MUST **also** enforce a finite ceiling on the octets it will buffer from an untrusted source and MUST refuse at that ceiling **without completing the parse** — same `ENVELOPE_TOO_LARGE` / 413. The ceiling is a deployment parameter, not a wire constant; it MUST NOT be set below 1 MiB (a lower ceiling could refuse a conformant envelope), and a deployment SHOULD publish it. This is what makes "at every write path, not only an HTTP body gate" implementable: each path bounds its own intake instead of delegating the bound to a body gate it may not have.

The two coincide on a conformant relay path. A relay that re-emits has already recomputed JCS to verify, and CC 2.6.1.1 requires it to re-canonicalize member order while holding member presence/absence fixed — so what it forwards is the canonical form, and there the received octets **are** the canonical octets. The gap exists only where a non-canonical serialization first enters. Consequently **the interoperable serialization is the canonical one**: a party that transmits an inflated serialization of a semantically-conformant envelope MAY have it refused by a receiver's intake ceiling, and that refusal is conformant receiver behavior, not a receiver defect.

2.6.1.4 Worked attack the rule closes

Without the rule (implicit / unspecified) failure mode:

> *Alice's CEG-Conforming Producer signs an envelope omitting `epistemic_mode`. Bob's CEG-Conforming Relay receives, parses, materializes the default `"direct"`, re-serializes via JCS, and forwards to Carol. Carol receives the relay-modified envelope with the new bytes, computes JCS, verifies signature →* ***FAILS*** *because Alice signed over bytes without the `epistemic_mode` member. Carol cannot distinguish "Bob is a malicious relay corrupting the bytes" from "Bob is honestly applying defaults to be helpful." Carol rejects. The attestation is lost in transit despite no party acting in bad faith.*

With the rule (explicit, normative):

> *Bob's relay MUST preserve member presence/absence exactly. Bob forwards the as-received bytes. Carol's verify succeeds. The semantic interpretation step at Carol (applying default `"direct"` to the absent member) happens AFTER signature verification.*

2.6.1.5 Verification flow (informational)

```
On signature verify:
    1. Receive envelope from wire as object O (preserved as-received)
       and signature S
    2. Compute canonical bytes B = JCS(O)
    3. Verify S over B via the hybrid Ed25519 + ML-DSA-65 path per S5.2.1
    4. If signature valid:
        a. Compute effective semantics by applying S4 defaults to absent
           optional fields (interpretation-time only)
        b. Apply consumer policy per S8 over the resulting semantic shape
    5. If forwarding/storing:
        a. Store/forward object O AS RECEIVED -- do not normalize, strip,
           or materialize defaults
        b. Forward original signature S unchanged
```

2.6.1.6 Scope of the canonicalization rule

Scope pointer: the JCS canonicalization rules above (JCS-as-encoding, omit-vs-materialize + relay-preservation, per-field catalog, worked attack).

What this rule does NOT do:

- Introduce a new encoding format — JCS is the only encoding
- Change any semantic interpretation — defaults still apply at interpretation time per CC 2.1
- Modify the datetime / hex / time / H3 sub-rules — those are domain-specific and compose under JCS
- Wire-break prior emissions — emissions that omitted optional fields remain valid; emissions that explicitly emitted defaults likewise remain valid; what the rule normatively prohibits is RELAY-TIME mutation of presence/absence

2.6.1.7 The signed object: member set, placement and types (normative)

Generated from `constitution/signed_object.tsv` by `tools/build_cc_signed_object.py`; the same data renders `manifests/signed_object.cddl` and `manifests/signed_object.json`, and `manifests/signed_object_vectors.json` holds the byte vectors. Edit the TSV.

The object (normative). The signed object of an attestation row is `attestation_envelope`: one JSON object. The signing input is JCS(`attestation_envelope`) (CC 2.6.1.3) and nothing else — the row's own signature members, every cosignature and every column beside the row are outside the bytes, so a cosignature is never inside the bytes it signs. Seven of those columns are mirrored *inside* the bytes under the member `row`, a closed set; a row with no mirror is refused `envelope_member_missing`, and a column that disagrees with its mirror is refused `typed_diverges_from_envelope`. The claim's instants ride inside the bytes as `asserted_at` and `expires_at`, and the columns are read back out of them. The row's own signature is the attester's: the classical half verifies against `attesting_key_id`'s pinned Ed25519 key over the signing input, the post-quantum half against its ML-DSA-65 key over the signing input followed by the raw Ed25519 signature ([hybrid-sig], CC 2.6.5); at the federation tier both halves verify or the row is refused. `original_content_hash` is the lowercase hex SHA-256 of the signing input and is recomputed at ingest. A second signer over the same bytes is an entry of `additional`

_scrubs[], beside the row, verified against its own key. A JCS image above 1 MiB is refused ENVELOPE_TOO_LARGE (CC 2.6.1.3).

Refusals at this door (normative). A shape refusal of the signed object is the token federation_malformed_row with one reason from this closed set: envelope_member_missing (the mirror or another required member is absent), envelope_member_invalid (a member is ill-typed or an instant is not in the CC 2.6.2 form), envelope_member_null (an optional member is null), typed_diverges_from_envelope (a column beside the row differs from its mirror), envelope_row_member_unknown (the row mirror carries a member outside its closed seven), content_hash_mismatch (original_content_hash is not the SHA-256 of the signing input) and subject_key_id_malformed (an element of subject_key_ids is neither a key_id nor the tagged canonical form of CC 2.3.2.1). A withdraws that meets no rule of CC 2.4.1.1 is refused federation_withdraws_not_admitted. A set out of order or with a duplicate is federation_set_semantics_unsorted, a type outside the five is attestation_type_unregistered, a failed signature half is SIGNATURE_VERIFICATION_FAILED, and an envelope over the cap is ENVELOPE_TOO_LARGE.

Placement vocabulary. signed:row — inside attestation_envelope.row. signed:envelope — inside attestation_envelope, typed by persist. signed:envelope-extra — inside attestation_envelope, typed by CC 2.1, carried untyped by persist and validated at the family's gate. beside — outside the bytes, replicated, bound to the bytes by a gate. beside:signature — the row's own signature members. beside:cosign — an additional_scrubs[] entry. local — persist-internal, not a wire contract. not-on-wire — named by this text, not carried on the wire. An array's kind is set (sorted by JCS string before signing) or sequence (as authored), per CC 2.6.1.1.1.

Inside the signed bytes — the mirror (closed, exactly seven)

Member	Placement	Type	Array	Presence	Default	CC	Note
row.attestation_id	signed:row	uuid	—	REQUIRED	—	2.6.1.7	The row's identity, minted by the stamper before the bytes exist (UUID v4); the composers' references_attestation_id points at it.
row.attesting_key_id	signed:row	key-id	—	REQUIRED	—	2.1 2.6.8	The signer. The row's own signature verifies against this key's pinned Ed25519 and ML-DSA-65 public keys; there is no separate signer slot inside the bytes.
row.attestation_type	signed:row	row-type	—	REQUIRED	—	2.4	One of the five, byte-compared; anything else is refused attestation_type_unregistered.
row.attested_key_id	signed:row	key-id	—	REQUIRED	the attester	2.1	Defaults to the attester at the mint (a self-claim).

Member	Placement	Type	Array	Presence	Default	CC	Note
row.subject_key_ids	signed:row	[* subject-id]	set	REQUIRED (may be empty)	[]	2.3 2.3.2.1 2.6.1.1.1	Always present in the mirror, an empty array when there is no subject; the column beside the row is absent when empty. An element is a key_id verbatim or the tagged canonical:{alg}:{hex} form; uppercase and whitespace are refused at emit. A set (2.6.1.1.1): sorted and unique, an unsorted or duplicated set refused federation_set_semantics_unsorted.
row.cohort_scope	signed:row	cohort-scope	—	REQUIRED	federation	2.3.3 5.2	Closed set of seven. The one mirror member a placement-touching re-sign may narrow; it may never be re-authored.
row.weight	signed:row	float	—	OPTIONAL	absent	2.1	Absent in the mirror iff the column is NULL. A non-finite value is refused at the stamp rather than serialised as null.

Inside the signed bytes — members persist types

Member	Placement	Type	Array	Presence	Default	CC	Note
row	signed: envelope	row-mirror	—	REQUIRED	—	2.6.1.7	The signed twin of the seven typed columns beside the row. The member set is closed (an eighth member is refused); a row with no mirror is refused envelope_member_missing, a mirror that disagrees with a column is refused typed_diverges_from_envelope.

Member	Placement	Type	Array	Presence	Default	CC	Note
`asserted_at`	signed: envelope	`rfc3339-ms`	—	REQUIRED	stamped at the mint	2.6.2 2.6.7	The claim's instant, truncated to the millisecond and rendered YYYY-MM-DDTHH:MM:SS.sssZ before signing; the column beside the row is read back out of this member and never from a second clock. Judged against the receiver's clock only where 2.6.7 says (the future bound; W on chain-judged rows).
`expires_at`	signed: envelope	`rfc3339-ms`	—	OPTIONAL	absent (no expiry)	2.1 3.1.3.2	The column's signed twin; present iff the column is set, any other pairing refused. REQUIRED on census: share:{class}: v1 and census: member:share: v1, where it is the period's end (census_return_ expired after it).
`dimension`	signed: envelope	`tstr`	—	REQUIRED on every row type	—	2.1 2.4 3.1 3. 1.7	The namespace prefix plus leaf, resolved through manifests/ namespace_ registry.json on all five row types (the job label on a delegates_to as much as the family on a scores).

Member	Placement	Type	Array	Presence	Default	CC	Note
`references_attestation_id`	signed: envelope	`uuid`	—	REQUIRED on supersedes, withdraws, recants	—	2.4.1 4.5.1.1	The composer target: the prior row's attestation_id. Its meaning is read only with the row type as discriminator (three readings, CC 4.5.1.1); a fourth reading is a separate member, which is why consent_supersedes exists.
`scope`	signed: envelope	`tstr / [* tstr]`	set	REQUIRED on consent:state: granted and on delegates_to	—	2.1 3.3.1 4. 4.3.4.3	The canonical form is the array: an emitter writes ["retain"], and a bare token is admitted as the one-element array it denotes; admission normalises nothing in the signed bytes.
`pre_rotation_commitment`	signed: envelope	`hex64`	—	OPTIONAL (charter)	absent	3.2	Pre-rotation commitment on a charter (the KERI lesson).
`recovers`	signed: envelope	`key-id`	—	REQUIRED on a charter recovery	absent	3.2	The predecessor root being rotated.
`successor_keys`	signed: envelope	`[* key-id]`	set	REQUIRED on a charter recovery	absent	3.2	The pre-committed successor key set.
`withdrawal_reason`	signed: envelope	`tstr`	—	OPTIONAL on withdraws	absent	2.4.1	The producer's stated reason.
`consent_supersedes`	signed: envelope	`uuid`	—	OPTIONAL on consent: state:*	absent	3.3.1 4.5.1.1	The prior consent statement this one replaces. Separate from references_attestation_id by the CC 4.5.1.1 op split.
`delivery_mode`	signed: envelope	`"pull" / "push"`	—	OPTIONAL	pull	2.1 2.3.3	Recipient-receive axis; the processor is edge's.
`deletion_window`	signed: envelope	`tstr`	—	OPTIONAL	absent	3.3.1	Deadline by which a consented payload must be deleted.

Member	Placement	Type	Array	Presence	Default	CC	Note
`listed`	signed: envelope	`"public"`	—	OPTIONAL	absent (private roster)	2.1 3.2	One admissible value; admitted only on a community membership signed by its own subject.
`media`	signed: envelope	`object`	—	OPTIONAL	absent	3.3.13	The multimedia Source struct, kept as JSON and validated at the door.
`widened_at`	signed: envelope	`rfc3339-ms`	—	REQUIRED on a widening supersedes, absent otherwise	absent	4.4.3.3.1 2. 6.7	When the widening itself was placed; asserted_at on a widening is the claim's own instant carried verbatim.
`accord_root`	signed: envelope	`key-id`	—	REQUIRED on accord:* rows	absent	4.2	Which accord an accord:* row belongs to; a foreign row carrying junk fails closed at the fold.
`attach_window_secs`	signed: envelope	`uint`	—	OPTIONAL on trust:charter:v1	absent (anchor only)	2.1 3.2	T4a attach window.
`witness_cadence_secs`	signed: envelope	`uint`	—	REQUIRED on a witnessed lineage's charter	absent	2.1 3.2	T6 cadence.
`witness_quorum`	signed: envelope	`uint`	—	REQUIRED on a witnessed lineage's charter	absent (= 0, witnessed mode off)	2.1 3.2	T6 quorum; floor(\
`attached_head_digest`	signed: envelope	`hex64`	—	REQUIRED on trust:accepts:v1	—	2.1 3.2	T4a: the lineage head the acceptance attaches on.
`recovery_commitments`	signed: envelope	`{* key-id => hex64}`	—	REQUIRED on the accord's charter	—	2.1 4.2.6	Holder key id to commitment; object keys sort under JCS like any member.

Member	Placement	Type	Array	Presence	Default	CC	Note
`for_key_id`	signed: envelope	`key-id`	—	REQUIRED on census:share: {class}: v1, census: member: share:v1	absent	2.1 3.1.3 6. 1.9	The addressed recipient. The row is held by the author's node and this key's node and by no other; the serve gate sends it to this key only, no widening applies, and any other node refuses it federation_ addressed_ not_recipient. Also carried on consent:state rows.
`period`	signed: envelope	`uint`	—	REQUIRED on census:share: {class}: v1, census: member: share:v1, census: partial: {class}: v1, census: member: partial: v1, census: composite: {class}:v1, ballot:tally: {poll_id}: v1, census: expected_ active:v1	absent	2.1 6.1.9	The census period index on the global cadence (CC 5.4.5).
`return_token`	signed: envelope	`hex64`	—	REQUIRED on census:share: {class}: v1, census: member: share:v1	absent	2.1 6.1.9	H(node_secret ‖ period): the per-period token a contributor later checks for inclusion; never a node id.
`share_index`	signed: envelope	`uint`	—	REQUIRED on census:share: {class}: v1, census: member: share:v1, census: partial: {class}: v1, census: member: partial:v1	absent	2.1 6.1.9	0 or 1: which of the two additive shares, and which aggregator's partial. Any other value is census_share_ index_invalid.

Member	Placement	Type	Array	Presence	Default	CC	Note
share_count	signed: envelope	uint	—	REQUIRED on census: partial: {class}: v1, census: member: partial:v1	absent	2.1 6.1.9	Shares the aggregator summed into this partial; both partials of a period must agree.
source_count	signed: envelope	uint	—	REQUIRED on census: composite: {class}:v1, ballot:tally: {poll_id}:v1	absent	2.1 6.1.2.1 6.1.9	Distinct owners behind the composite; below 10 the cell is refused census_composite_below_k.
n_eff	signed: envelope	uint	—	REQUIRED on census: composite: {class}:v1, ballot:tally: {poll_id}:v1	absent	2.1 6.1.2.1.2 6.1.9	The effective source count; equals source_count by construction for one-hot returns; aggregation_meta_dominated below half of source_count.
max_source_multiplicity	signed: envelope	uint	—	REQUIRED on census: composite: {class}:v1, ballot:tally: {poll_id}:v1	absent	2.1 6.1.2.1 6.1.9	The largest multiplicity cluster; aggregation_meta_multiplicity above the floor.
mass_commitment_hex	signed: envelope	hex64	—	REQUIRED on census: composite: {class}:v1, ballot:tally: {poll_id}:v1	absent	2.1 6.1.2.1 6.1.9	Merkle root over the per-member masses.
member_commitment_hex	signed: envelope	hex64	—	REQUIRED on census: partial: {class}: v1, census: member: partial: v1, census: composite: {class}:v1, ballot:tally: {poll_id}: v1, census: expected_active:v1	absent	2.1 6.1.2.1.1 6.1.9	Merkle root over the accepted tokens; a contributor checks its token's path against it; both partials of a period must carry the same root.

Member	Placement	Type	Array	Presence	Default	CC	Note
mass_total	signed: envelope	tstr	—	REQUIRED on census: partial: {class}: v1, census: member: partial: v1, census: composite: {class}:v1, ballot:tally: {poll_id}:v1	absent	2.1 6.1.9	The decimal string of a field element: the sum of the cells. On a composite it equals source_count and the sum of the two partials' mass_total, else census_composite_mass_total_mismatch / census_composite_partials_mismatch.
aggregation_algorithm_id	signed: envelope	tstr	—	REQUIRED on census: partial: {class}: v1, census: member: partial: v1, census: composite: {class}:v1, ballot:tally: {poll_id}:v1	absent	2.1 6.1.2.1 6.1.9	Names the field prime, the one-hot bucket layout and the proof system the shares were made under.
noise_floor_descriptor	signed: envelope	tstr	—	OPTIONAL on census: composite: {class}:v1, ballot:tally: {poll_id}:v1	absent	2.1 6.1.2.1 6.1.9	What survives below the floor, when a cell is withheld.
version	signed: envelope	uint	—	REQUIRED on census: composite: {class}:v1, ballot:tally: {poll_id}:v1	absent	2.1 6.1.2.1 6.1.9	3: the composite carries the CC 6.1.2.1 verify inputs as signed members; a lower value is refused as the tier object's would be.
partial_a	signed: envelope	uuid	—	REQUIRED on census: composite: {class}:v1, ballot:tally: {poll_id}:v1	absent	2.1 6.1.9	The attestation_id of the share-index-0 partial this composite sums. references_attestation_id stays unset on a composite.
partial_b	signed: envelope	uuid	—	REQUIRED on census: composite: {class}:v1, ballot:tally: {poll_id}:v1	absent	2.1 6.1.9	The attestation_id of the share-index-1 partial.

Member	Placement	Type	Array	Presence	Default	CC	Note
`window_periods`	signed: envelope	`uint`	—	REQUIRED on census: expected_active:v1	absent	2.1 4.5.13 6.1.9	4: a member is expected-active when active in at least one of the last four periods.
`active_count`	signed: envelope	`uint`	—	REQUIRED on census: expected_active:v1	absent	2.1 4.5.13 6.1.9	Owners active within the window, after the ratchet; every member recomputes it, else census_expected_active_mismatch.
`floor_applied`	signed: envelope	`bool`	—	REQUIRED on census: expected_active:v1	absent	2.1 4.5.13 6.1.9	Whether the ratchet's floor, not the raw count, set active_count this period.

Inside the signed bytes — members this text types and persist carries untyped

Member	Placement	Type	Array	Presence	Default	CC	Note
`score`	signed: envelope-extra	`float`	—	REQUIRED on scores	—	2.1 3.1.7	In [-1, 1]; the family's polarity class is an admission rule (CLM-polarity-admission).
`confidence`	signed: envelope-extra	`float`	—	REQUIRED on scores	—	2.1	In [0, 1].
`context`	signed: envelope-extra	`tstr`	—	OPTIONAL	absent	2.1	Free form; not parsed by the substrate.
`evidence_refs`	signed: envelope-extra	`[* tstr]`	sequence	OPTIONAL (some families require it)	absent	2.1 2.6.1.1.1 2.6.1.3	Producer-asserted order, preserved. Above the 1 MiB cap the payload moves to the fountain plane and is named here by content hash.
`valid_until`	signed: envelope-extra	`rfc3339-ms`	—	OPTIONAL	absent	2.1 2.6.2	Consumer-policy staleness, independent of expires_at.

Member	Placement	Type	Array	Presence	Default	CC	Note
epistemic_mode	signed: envelope-extra	"direct" / "crypto" / "hearsay" / "derivative" / "appeal"	—	OPTIONAL	direct	2.1 2.5 2.6.1.2	Interpretation-time default; an explicit default value is signed as written (2.6.1.1).
witness_relation	signed: envelope-extra	"self" / "external" / "derived"	—	OPTIONAL	external	2.1 2.6.1.2	
oversight_mode	signed: envelope-extra	"HITL" / "HOTL" / "HOOTL"	—	OPTIONAL	absent (per-cell default)	2.1 2.6.1.2	CC 2.1 writes the default as null, which means absent; an explicit JSON null on any optional member is refused federation_malformed_row / envelope_member_null (CC 2.6.1.1.1).
occurrence_id	signed: envelope-extra	tstr	—	OPTIONAL	absent (read as occurrence-0)	2.1 2.6.1.2	Self-asserted (CC 8.3.1).
occurrence_count	signed: envelope-extra	uint	—	OPTIONAL	absent (read as 1)	2.1 2.6.1.2	At least 1.
occurrence_role	signed: envelope-extra	"primary" / "shared" / "replica"	—	OPTIONAL	absent (read as primary)	2.1 2.6.1.2	
stake	signed: envelope-extra	"free" / "reputational" / "capital" / "cryptoeconomic"	—	OPTIONAL	reputational	2.1 2.5	
community_id	signed: envelope-extra	key-id	—	REQUIRED iff cohort_scope is community or affiliations	—	2.1 2.3.3 4.4.3.2.8	The community whose roster gates visibility; refused when missing at those scopes.
family_id	signed: envelope-extra	key-id	—	REQUIRED iff cohort_scope is family	—	2.1 2.3.3 3.3.4	
session_id	signed: envelope-extra	tstr	—	REQUIRED on session:*	—	2.1 3.1.3.1	Verbatim, case-preserved.
claimed_at	signed: envelope-extra	rfc3339-ms	—	REQUIRED on session:*	—	2.1 3.1.3.1	
custody_state	signed: envelope-extra	"here" / "none"	—	REQUIRED on custody:ack	—	2.1 3.1.3.3	Refused custody_ack_malformed otherwise.

Member	Placement	Type	Array	Presence	Default	CC	Note
`group_kind`	signed: envelope-extra	`"family" / "community"`	—	REQUIRED on membership: proposal	—	2.1 3.1.3.2	
`role`	signed: envelope-extra	`tstr`	—	OPTIONAL on membership: proposal, restated on membership: acceptance	absent	2.1 3.1.3.2	
`proposal_hash`	signed: envelope-extra	`hex64`	—	REQUIRED on membership: acceptance and membership: decline	—	2.1 3.1.3.2	The answered proposal's original_ content_ hash.
`subject_sha256_ed25519_raw`	signed: envelope-extra	`hex64`	—	OPTIONAL on moderation:*	absent	2.1 2.4.1.1	Refused moderation_ subject_ digest_ malformed when malformed.
`halt_fuse_secs`	signed: envelope-extra	`uint`	—	OPTIONAL on the accord's charter	86400 (a constant of this document)	2.1 4.2.1.1	The halt latch reads the constant when the member is absent.
`witnesses`	signed: envelope-extra	`[* key-record]`	set	REQUIRED on a witnessed lineage's charter	—	2.1 3.2	A set, sorted by each element's JCS string (the 2.6.1.1.1 default made explicit).
`history_on_join`	signed: envelope-extra	`"full" / "from_join"`	—	OPTIONAL	from_join	2.1 4.5.12.1	
`supersession_reason`	signed: envelope-extra	`tstr`	—	OPTIONAL on supersedes	absent	2.4.1	
`differs_in`	signed: envelope-extra	`[* tstr]`	set	OPTIONAL on supersedes; a widening lists cohort_scope	absent	2.4.1 4.4.3.3.1	Member names that differ from the prior. Array kind not declared in 2.4.1, so the 2.6.1.1.1 default (set, sorted) applies.
`recantation_reason`	signed: envelope-extra	`tstr`	—	OPTIONAL on recants	absent	2.4.1	
`what_was_false`	signed: envelope-extra	`tstr`	—	OPTIONAL on recants	absent	2.4.1	

Member	Placement	Type	Array	Presence	Default	CC	Note
`delegation_purpose`	signed: envelope-extra	`tstr`	—	OPTIONAL on delegates_to	absent	2.4.1 2.4.1.2	owner_binding is the one single-valued purpose.
`delegation_valid_from`	signed: envelope-extra	`rfc3339-ms`	—	OPTIONAL on delegates_to	the edge's asserted_at	2.4.1	
`delegation_valid_until`	signed: envelope-extra	`rfc3339-ms`	—	REQUIRED on a termed delegates_to (CC 2.4.1), otherwise OPTIONAL	absent (open-ended)	2.4.1 4.4.3.2.8	At most 365 days after delegation_valid_from; refused delegation_term_required.
`*`	signed: envelope-extra	`any`	—	OPTIONAL	—	2.1.1 3.3	Every other member, including subject_kind and the per-family payloads of CC 3.3, rides the envelope untyped by the substrate and is preserved on relay (2.1.1); families validate their own at their gates. This table does not enumerate them.
`consent_basis`	signed: envelope-extra	`[* consent-basis]`	sequence	OPTIONAL (capacity:*, observation: access_assumption: v1)	absent	2.1 3.3.1	The grants a witness relied on.
`n_eff_pr`	signed: envelope-extra	`number`	—	OPTIONAL (carried on capacity:*; validated at the capacity:* gate)	absent	2.1	Participation-ratio effective sample size; gated.
`n_eff_h`	signed: envelope-extra	`number`	—	OPTIONAL (carried on capacity:*; validated at the capacity:* gate)	absent	2.1	Entropy-perplexity effective sample size; carried for audit.

Member	Placement	Type	Array	Presence	Default	CC	Note
`sample_size`	signed: envelope-extra	`uint`	—	OPTIONAL (carried on capacity:*; validated at the capacity:* gate)	absent	2.1	Rows in the scoring matrix.
`feature_dim`	signed: envelope-extra	`uint`	—	OPTIONAL (carried on capacity:*; validated at the capacity:* gate)	absent	2.1	Features per row.
`sample_size_gate`	signed: envelope-extra	`uint`	—	OPTIONAL (carried on capacity:*; validated at the capacity:* gate)	absent	2.1	Configured minimum sample size.
`target_n_eff`	signed: envelope-extra	`number`	—	OPTIONAL (carried on capacity:*; validated at the capacity:* gate)	absent	2.1	Configured target effective sample size.
`min_feature_dim`	signed: envelope-extra	`uint`	—	OPTIONAL (carried on capacity:*; validated at the capacity:* gate)	absent	2.1	Configured minimum feature dimension.
`n_eff_floor`	signed: envelope-extra	`number`	—	OPTIONAL (carried on capacity:*; validated at the capacity:* gate)	absent	2.1	Configured effective-sample-size floor.
`window`	signed: envelope-extra	`uint`	—	OPTIONAL (capacity:*, observation: access_assumption: v1)	absent	2.1	The scoring or observation window, in rows.
`trace_level`	signed: envelope-extra	`"generic" / "detailed" / "full_traces"`	—	REQUIRED on observation: access_assumption: v1	absent	2.1 3.1.9.4	The trace level the observed agent's install emitted at.
`occurrence_key_ids`	signed: envelope-extra	`[* key-id]`	set	REQUIRED on observation: access_assumption: v1	absent	2.1 3.1.9.4	The occurrences whose rows the witness read.

Member	Placement	Type	Array	Presence	Default	CC	Note
window_from	signed: envelope-extra	rfc3339-ms	—	REQUIRED on observation: access_assumption: v1	absent	2.1 2.6.7 3.1.9.4	Start of the observation window.
window_to	signed: envelope-extra	rfc3339-ms	—	REQUIRED on observation: access_assumption: v1	absent	2.1 2.6.7 3.1.9.4	End of the observation window.
channel_classes	signed: envelope-extra	[* tstr]	set	REQUIRED on observation: access_assumption: v1	absent	2.1 3.1.9.4	Channel classes the witness saw the subject through, as the emitter names them.
amendment_digest	signed: envelope-extra	hex64	—	REQUIRED on identity_amendment: v1	absent	2.1 3.1.5	SHA-256 of the applied identity amendment.
authorized_by	signed: envelope-extra	uuid	—	REQUIRED on identity_amendment: v1	absent	2.1 3.1.5 4.3	The deferral: response: approved: v1 row that authorised it; admission checks it.

Beside the row — outside the bytes, bound to them

Member	Placement	Type	Array	Presence	Default	CC	Note
attestation_id	beside	uuid	—	REQUIRED	—	2.6.1.7	Equals row.attestation_id or the row is refused.
attesting_key_id	beside	key-id	—	REQUIRED	—	2.1	Equals row.attesting_key_id.
attested_key_id	beside	key-id	—	REQUIRED	—	2.1	Equals row.attested_key_id.
attestation_type	beside	row-type	—	REQUIRED	—	2.4	Equals row.attestation_type.
weight	beside	float	—	OPTIONAL	absent	2.1	Equals row.weight; absent iff absent.
asserted_at	beside	rfc3339	—	REQUIRED	—	2.6.7	Read out of the signed member at the mint; a column that disagrees is refused typed_diverges_from_envelope.

Member	Placement	Type	Array	Presence	Default	CC	Note
`expires_at`	beside	`rfc3339`	—	OPTIONAL	absent	2.1	Present iff the signed member is.
`subject_key_ids`	beside	`[* subject-id]`	set	OPTIONAL (absent when empty)	absent	2.3	Equals row. subject_key_ids as a list.
`cohort_scope`	beside	`cohort-scope`	—	REQUIRED	federation	2.3.3	Equals row. cohort_scope.
`attestation_envelope`	beside	`envelope`	—	REQUIRED	—	2.1 2.6.1 2.6.1.3	The signed object itself, stored and relayed as received (2.6.1.1). The signing input is JCS(attestation_envelope) and nothing else.
`original_content_hash`	beside	`hex64`	—	REQUIRED	—	2.6.1.3 2.6.3	Lowercase hex SHA-256 of JCS(attestation_envelope); recomputed at federation-tier ingest and refused on mismatch.
`additional_scrubs`	beside	`[* scrub-sig]`	sequence	OPTIONAL (absent when empty)	absent	2.4 3.2	Second and later signers over the same JCS bytes, each verified against its own scrub_key_id's pinned keys. This is the carrier of quorum on a row.
`persist_row_hash`	beside	`hex64`	—	OPTIONAL (store-computed; absent on a row as emitted)	—	—	Store-computed over the row's own serialisation (not JCS), excluding itself; informational on the wire, never re-verified on read (CC 2.6.1.3). Not a conformance target.
`withdraws_admission_rule`	local	`uint`	—	OPTIONAL	absent	2.4.1.1	Audit metadata: which rule (0 to 4) admitted a withdraws; persist-internal.

Member	Placement	Type	Array	Presence	Default	CC	Note
`tier`	local	`"local" / "federation"`	—	OPTIONAL	federation	3.1.3	Persist-internal; absent on the wire for a federation row. Not in the signed bytes, so a promoted row is byte-identical to a native one.
`promoted_at`	local	`rfc3339`	—	OPTIONAL	absent	3.1.3	Persist-internal.
`scrub_signature_classical`	beside: signature	`base64`	—	REQUIRED	—	2.6.5 2.6.1.1.1 5.3.2.4.3.1	Ed25519 (RFC 8032) over JCS(attestation_envelope); RFC 4648 section 4 standard alphabet, padded.
`scrub_signature_pqc`	beside: signature	`base64`	—	REQUIRED at the federation tier	—	2.6.5 5.3.2.4.3.1 6.1.3	ML-DSA-65 (FIPS 204) over JCS(attestation_envelope) followed by the raw 64 Ed25519 signature bytes (the bound-payload form); verified under the strict hybrid policy, never classical-only.
`scrub_key_id`	beside: signature	`key-id`	—	REQUIRED	—	2.1	On an attestation row the verifier resolves attesting_key_id, so this names the same key; assemble sets both from one key. (On a key record it may name an anchor, CC 3.2.)
`scrub_timestamp`	beside: signature	`rfc3339`	—	REQUIRED	—	2.6.7	When the signature was issued: asserted_at, or widened_at on a widening. Outside the bytes.
`pqc_completed_at`	beside: signature	`rfc3339`	—	OPTIONAL	—	5.3.2.4.3.1	When the ML-DSA-65 half was attached. Outside the bytes.
`additional_scrubs[].scrub_key_id`	beside: cosign	`key-id`	—	REQUIRED	—	2.4 3.2	The cosigner; must resolve to a registered key.

Member	Placement	Type	Array	Presence	Default	CC	Note
additional_scrubs[].scrub_signature_classical	beside: cosign	base64	—	REQUIRED	—	2.6.5	Ed25519 over the same JCS(attestation_envelope).
additional_scrubs[].scrub_signature_pqc	beside: cosign	base64	—	REQUIRED at the federation tier	—	2.6.5	ML-DSA-65 over the same JCS bytes followed by this entry's Ed25519 signature.
additional_scrubs[].cosigned_at	beside: cosign	rfc3339-ms	—	OPTIONAL	absent	2.6.7	When the cosignature was applied; outside every preimage, so stamping it moves no signed byte.

The key record — the second signed object (outer shape; its envelope's member table is the key plane's)

Member	Placement	Type	Array	Presence	Default	CC	Note
key_id	beside	key-id	—	REQUIRED	—	2.6.8 3.2	<label>-<fingerprint>, the fingerprint the first ten lowercase base32 characters of sha256(ed25519 public key).
pubkey_ed25519_base64	beside	base64	—	REQUIRED	—	3.2 5.3.2.4.3.1	Raw 32-byte key.
pubkey_ml_dsa_65_base64	beside	base64	—	REQUIRED for algorithm hybrid	—	5.3.2.4.3.1	Raw 1952-byte key; the substrate takes writes only at algorithm hybrid.
algorithm	beside	tstr	—	REQUIRED	—	5.3.2.4.3.1	hybrid on every admitted record.
identity_type	beside	tstr	—	REQUIRED	—	3.4.7.1	A set rendered as one string; see CC 3.4.7.1.
identity_ref	beside	tstr	—	REQUIRED	—	3.2	
valid_from	beside	rfc3339	—	REQUIRED	—	2.1	
valid_until	beside	rfc3339	—	OPTIONAL	absent	2.1	Its signed twin is inside registration_envelope (CLM-key-validity-window).

Member	Placement	Type	Array	Presence	Default	CC	Note
registration_envelope	beside	object	—	REQUIRED	—	3.2 3.3.6	The key record's signed object: JCS(registration_envelope) is the signing input for the base scrub and every additional scrub. Its member table is the key plane's (key: record:v1) and is not enumerated here.
original_content_hash	beside	hex64	—	REQUIRED	—	2.6.1.3	Lowercase hex SHA-256 of JCS(registration_envelope).
scrub_signature_classical	beside: signature	base64	—	REQUIRED	—	2.6.5	Ed25519 over JCS(registration_envelope).
scrub_signature_pqc	beside: signature	base64	—	REQUIRED once hybrid-complete	—	2.6.5	ML-DSA-65 over JCS(registration_envelope) followed by the Ed25519 signature bytes.
scrub_key_id	beside: signature	key-id	—	REQUIRED	—	3.2	Self-scrub when equal to key_id; an anchor scrub when it names a pinned anchor (the one record kind where the scrubber may differ from the subject).
scrub_timestamp	beside: signature	rfc3339	—	REQUIRED	—	2.6.7	
pqc_completed_at	beside: signature	rfc3339	—	OPTIONAL	—	5.3.2.4.3.1	
persist_row_hash	beside	hex64	—	REQUIRED	—	—	Store-computed; excludes itself and consent_role; informational.
roles	beside	[* tstr]	set	OPTIONAL	[]	3.4.7.1	Serialised as roles; the record also carries roles inside the signed registration_envelope.
attestation_evidence	beside	object	—	OPTIONAL	absent	4.2.2	Hardware attestation blob.
consent_role	local	tstr	—	OPTIONAL	absent	3.4.7.2	Mutable operational marker, excluded from persist_row_ hash and from the signed bytes.
additional_scrubs	beside	[* scrub-sig]	sequence	OPTIONAL (absent when empty)	absent	3.2	Second and later scrubs over the same JCS(registration_envelope); what lets a replicated record prove its own m-of-n.

Rules this table carries. A set-kind array is sorted **and** unique, refused otherwise as `federation_set_semantics_unsorted` with the member named; a relay never re-sorts it, since re-sorting would move signed bytes. The canonical `scope` is the array; a bare token is admitted as the one-element array it denotes, and nothing is normalised in signed bytes. An explicit JSON `null` on an optional member is refused as malformed (CC 2.6.1.1.1). `witnesses[]` is a set. The quorum carrier is `additional_scrubs[]` beside the row, as CC 2.1 and CC 2.4 name it; the name `cosignatures[]` is retired.

Vectors. `manifests/signed_object_vectors.json` carries two test identities (Ed25519 and ML-DSA-65 seeds, public keys and the CC 2.6.8 `key_id` derived from each), and for each vector the envelope, its JCS bytes, `original_content_hash`, both signature halves and the assembled row: a minimal `scores`; the same claim with an explicit default (different bytes, both verify, CC 2.6.1.1); a cosigned variant; a `withdraws`; the at-cap and cap-plus-one envelopes as a recipe with their hashes; and refusal cases with their tokens. Signatures were produced and verified by the reference implementation (CIRISVerify); `tools/build_cc_signed_object.py -self-test` recomputes the JCS bytes, the hashes and the Ed25519 half in CI. A second implementation that reproduces these bytes and signatures has met the CC 2.6.1.1.1 byte-identity target for this object.

2.6.2 Date-time canonicalization

Every ISO 8601 / RFC 3339 datetime in this specification MUST be:

- UTC (suffix: literal `Z`; the offset form `+00:00` MUST NOT be used)
- Millisecond-precision (exactly three digits of fractional seconds; trailing zeros required)
- Lowercase `z` MUST NOT be used; uppercase `Z` only

Canonical form: `YYYY-MM-DDTHH:MM:SS.sssZ`. Example: `2026-05-28T13:45:09.000Z`. Producers MUST emit this form; consumers MUST reject any other form when verifying a signature.

2.6.3 Hexadecimal canonicalization

Every hex string used in canonical-bytes encoding (e.g., SHA-256 digests in `root_hash`, public-key fingerprints) MUST be **lowercase**, **unpadded** (no leading `0x`, no separators), and **byte-length-exact** (a SHA-256 digest is exactly 64 hex characters). Producers MUST emit lowercase; consumers MUST reject uppercase when verifying.

2.6.4 Versioning policy

The grammar can evolve only if every change announces whether it breaks the wire. SemVer makes that announcement machine-legible, so a peer knows from the version alone whether it can still interoperate.

CEG follows **SemVer 2.0.0** with these mapping rules:

- **MAJOR (X.0.0)** — any wire-incompatible change: removal of an envelope field, change of a field's semantic, removal of a structural primitive, change to canonical-bytes domain-separation labels, removal or breaking-redefinition of a CC 3.1 prefix, change to a CC 3.4 reservation, or change to the CC 2.6.9 / CC 2.2 conformance language.

- **MINOR (0.X.0)** — wire-compatible additions: new prefix in CC 3.1, new envelope field with documented default, new composition policy in CC 4.4, new endpoint shape in CC 5.3, new optional conformance subsection. Existing Conforming Producers and Consumers continue to interoperate without modification.
- **PATCH (0.0.X)** — clarifications, editorial fixes, additions to non-normative sections (open-vocabulary lists, glossaries CC 8.1), addition to CC 8.3 acknowledged-gaps, fixes to non-normative examples in CC 8.1.

The 0.x series indicates this specification is a Public Working Draft. Any 0.x → 0.(x+1) bump MAY include wire-breaking changes; consumers MUST treat 0.x as unstable until 1.0 publication. Once 1.0 is published, the rules above bind strictly.

A **deprecation** is announced by adding a `**DEPRECATED in 0.X**` marker to the affected element with a stated removal target (e.g., `removal: 1.2`). Deprecated elements MUST remain interoperable until the announced removal version. Removal in MAJOR or 0.MINOR per the rules above.

The wire vocabulary is a hash-pinned artifact. The set of message types the substrate recognizes — the *wire vocabulary* — is not the emergent product of individual repos' needs but a ratified contract, expressed as a versioned, hash-addressable artifact, `manifests/WIRE_VOCABULARY.md`, governed in **two tiers** (the RFC 8126 partitioned-registry discipline, the pattern proven by Nostr kind-ranges and Matrix reverse-DNS namespacing). **Tier 1** — message types whose canonicalization is load-bearing to an ethical primitive (attestations, votes, moderation/slashing, deferral, steward directives, key registration, the humanity-accord path, `withdraws`, goals, build provenance, content/blob fetch) — is closed and CC-ratified: a Tier-1 addition rides the ordinary CC 4.5.1 amendment (the "Standards Action" bar), a working reference implementation, and a coordinated cut. **Tier 2** — app-tier remote-procedure channels whose inner semantics are the app's concern — uses three opaque envelopes (`OpaqueRequest` / `OpaqueResponse` / `OpaqueEvent`) carrying a `kind` from a reserved per-repo range delegated to the range steward (the "Private Use" bar); no CC amendment is required to define a `kind`. Every ratifying repo **pins the artifact's SHA-256 at build**; a hash mismatch at cohabitation is a substrate-tier build failure, not a warning. Amendment is a substrate-coordinated event across the covenant of ratifying repos; unanimous hash-commit blocks the *cut*, never the *decision* (the decision is the CC 4.5.1 amendment). This governs the **transport vocabulary** — a surface distinct from, and complementary to, the frozen CC 1.7 1+4 attestation grammar: 1+4 freezes what a claim *is*; the wire vocabulary governs the message types that *carry* claims.

2.6.5 Normative References

The following documents are normatively cited; implementations MUST conform to them where referenced inline.

Short name	Normative document
[BCP 14]	RFC 2119 + RFC 8174 — keywords for use in RFCs
[FIPS-180-4]	FIPS 180-4 — SHA-256 and the SHA-2 family
[FIPS-202]	FIPS 202 — SHA-3 / SHAKE128 / SHAKE256 / TupleHash
[FIPS-204]	FIPS 204 — ML-DSA (Module-Lattice-Based Digital Signature Algorithm); CEG uses parameter set ML-DSA-65
[RFC-3339]	RFC 3339 — Date and Time on the Internet, with fractional-seconds disambiguation in CC 2.6.2 below

Short name	Normative document
[RFC-5905]	RFC 5905 — Network Time Protocol Version 4
[RFC-6962]	RFC 6962 — Certificate Transparency; this spec's transparency-log discipline tracks 6962 except where 6962-bis (RFC 9162) supersedes
[RFC-8032]	RFC 8032 — Edwards-Curve Digital Signature Algorithm (EdDSA); specifically Ed25519
[RFC-8174]	RFC 8174 — Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words (with BCP 14)
[RFC-8785]	RFC 8785 — JSON Canonicalization Scheme (JCS); used where this spec serializes JSON for signing
[RFC-9162]	RFC 9162 — Certificate Transparency v2.0 (CT-bis); MUST be used for new transparency-log integrations; older 6962 instances continue to interoperate
[ISO-639-1]	ISO 639-1:2002 — Codes for the representation of names of languages, two-letter
[BCP-47]	BCP 47 (RFC 5646) — Tags for identifying languages; for locale strings richer than ISO 639-1 alone
[RFC-9420]	RFC 9420 — Messaging Layer Security (MLS); the streaming epoch-key rekey (CC 5.1) conforms to MLS TreeKEM
[SFrame]	draft-ietf-sframe — Secure Frames; the per-frame AEAD chunk seal (CC 5.3.3.1) conforms to the SFrame model
[FIPS-203]	FIPS 203 — ML-KEM (Module-Lattice KEM); the post-quantum half of the hybrid KEM (ML-KEM-768)
[FIPS-204]	FIPS 204 — ML-DSA (Module-Lattice signatures); the post-quantum half of the hybrid signature (ML-DSA-65)
[RFC-9180]	RFC 9180 — Hybrid Public Key Encryption (HPKE); the `key_grant` wrap shape
[hybrid-sig]	The **bound-hybrid signature**: Ed25519 ([RFC-8032]) over the canonical bytes, and ML-DSA-65 ([FIPS-204]) over the canonical bytes ‖ `ed25519_sig` (the bound-payload form, CC 3.1.2.1); both halves verified at every federation-tier admission gate and on the store path, never an operator toggle — normative at CC 5.3.2.4.3. 1 and CC 6.1.3

Informational citations (Magnifica Humanitas, anthropological literature, Ubuntu philosophical literature, etc.) appear in CC 8.6.1 without normative force.

2.6.6 H3 cell canonicalization

Geographic claims compose with CC 3.3.3 `location_proof` and CC 3.2 `community` with `cohort_subkind: geographic`.

Geographic primitives in CEG use H3 hierarchical hexagonal indexing as the canonical cell-identifier encoding. H3 partitions the Earth's surface into hexagonal cells at 16 resolution levels (0 = coarsest, ~4.3 M km^2 per cell; 15 = finest, ~0.9 m^2 per cell). Each cell has a 64-bit integer ID, conventionally encoded as a 15-character lowercase hex string.

Canonical form for `cell_id`:

- 15-character lowercase hex string (no `0x` prefix; per CC 2.6.3)
- The cell encodes its own resolution in the standard H3 index bit layout; a conformant decoder extracts the resolution **via the H3 library** (the resolution field lives in bits 52–55), **NOT** by reading the high 4 bits — the high nibble is the H3 **mode marker** (cell-mode = `1`), not the resolution
- Leading zeros preserved (a resolution-0 cell at base position 0 is `8001fffffffffff`, not `1fffffffffff` — the high nibble `8` is the mode marker, correctly consistent with res-0)

Canonical form for `cell_resolution`:

- Integer in `[0, 15]`
- MUST equal the resolution **decoded from the H3 index** of `cell_id` (substrate verifies the redundancy on admission via the H3 library; mismatched pairs MUST be rejected as malformed)

2.6.6.1 Rough-only enforcement for `location_proof` (normative)

Location disclosure is bounded by the protocol itself, not by operator goodwill — the Non-maleficence principle made mechanical. A `location_proof` subject_kind Contribution (CC 3.3.3) MUST carry `cell_resolution` $\leq$ `7` (H3 resolution 7 hexagons average ~5 km^2 edge-length, sufficient for city/borough-level disclosure without block/building precision). Producers attempting to emit finer-resolution `location_proof` Contributions MUST have admission rejected at the substrate gate.

This is the wire-format-enforced privacy promise: **rough is rough by protocol**, not by operator policy. A producer cannot accidentally over-share at the protocol layer; a malformed client cannot publish a precise location even if its UI fails to gate. Substrate emits `hard_case:location_proof_resolution_violation` (CC 3.4.2) on rejection so operators can observe malformed-producer patterns.

2.6.6.2 Cell containment

A cell `C` at resolution `R_C` is **contained within** a cell `B` at resolution `R_B` iff:

- `R_C >= R_B` (the contained cell is at equal or finer resolution); AND
- The parent-walk from `C` at resolution `R_C - 1, R_C - 2,..., R_B` reaches exactly `B` (standard H3 hierarchy semantics; `h3ToParent` library call)

Used by community admission gates per CC 4.4.3.2 Policy M: a `location_proof` Contribution's `cell_id` MUST be contained within the geographic community's `geographic_constraint.cell_id` for membership admission.

2.6.6.3 Scope of the H3 rule

Scope pointer: the H3 canonicalization rules above (lowercase-hex `cell_id`, rough-only `cell_resolution` $\leq$ `7`, containment for community admission).

What the H3 rule does NOT do:

- Mandate H3 over alternative geospatial systems (S2, Geohash) — H3 is chosen for hex-cell uniformity, well-defined parent/child hierarchy, and protocol-agnostic absence of a centralized gazetteer dependency. Operator-internal use of other systems is unconstrained; the wire format uses H3.
- Provide a place-name registry. Communities self-name; cell IDs are the substrate-level binding. UI may map cell IDs to human-readable names per consumer policy.

- Restrict community-side `geographic_constraint.cell_resolution`. A community CAN scope itself at any resolution (e.g., an "Austin metro" community at resolution 5, ~250 km^2; a smaller-scale "Downtown Austin" community at resolution 7, ~5 km^2). The rough-only invariant applies to `location_proof`, NOT to community definitions.

2.6.7 Time and clocks

Every `signed_at`, `asserted_at`, `valid_until`, `delegation_valid_from`, `delegation_valid_until`, and `cosigned_at` in this specification refers to **wall-clock UTC** at the asserting peer's clock. Producers SHOULD synchronize via NTPv4 (RFC 5905) or Roughtime to a known-good time source. The maximum tolerated skew between attester clock and consumer clock for a freshness check is ±**5 minutes** by default; tighter thresholds MAY be applied by per-application consumer policy. Consumers receiving an attestation with `signed_at` more than 5 minutes in the future MUST reject as malformed.

Time-skew between cosigners on a single STH (CC 5.3.1) is bounded by the STH's own `signed_at` field; cosignatures with `signed_at` farther than 5 minutes from the STH's published `signed_at` MUST be rejected.

For long-lived attestations carrying `valid_until` in the future, the freshness check is "the attestation has not yet reached its `valid_until`, AND the current consumer clock is within ±5 minutes of the substrate's network-consensus clock"; a consumer whose clock drifts past the skew bound MUST fail-secure (reject) rather than accept.

The admission freshness window `W` (normative). Every row whose admission walks a `delegates_to` chain — a key wrap (`key_grant:*`), a consent row, a licence (`licensure:*`), the epoch-axis membership check — is judged at its signed `asserted_at` (CC 2.4.1.2.1); that instant is signer-chosen, so it is bounded. The substrate MUST refuse such a row whose `asserted_at` precedes the receiver's clock by more than **`W` = 72 hours** (259 200 s — the CC 4.2.6 participation-window baseline, one "until everyone has heard" constant), with the refusal token **`asserted_at_stale`**: the row is well-formed, not malformed, and it is not admitted-as-stale; the remedy is to re-sign, since a signer back from a week offline is present and can. Without the bound a delegate whose chain was withdrawn could sign a row dated inside its old live window and have it admit; with it, a withdrawn delegate keeps at most `W` of backdatable authority at a node that holds the withdrawal, which is exactly the honest divergence at a node that does not — a withdrawal takes full effect everywhere within `W`, no worse than replication already is. Rows not judged on a chain (traces, scores on the base) are not bound by `W`; the future bound above is unchanged.

2.6.8 FedCode: the kind-tagged identity shorthand; NodeCode v1 retained as `kind: node` (normative)

Federation `key_id`s are long opaque identifiers, unfit for a human to type or read aloud. **FedCode** is the **one** human-shareable shorthand — a compact, QR-able, checksummed render of a federation entity's identity for **bootstrap UX** — pinned here so **every implementation renders and parses the same code for the same key**, a cross-impl determinism requirement of the same class as CC 2.6.3 hex / CC 2.6.1 JCS. It is a deterministic render of an existing `key_id`, **not** a new envelope field — additive on the frozen 1+4 surface; the `kind` byte is a payload discriminator of the same discipline as `subject_kind` riding the single `scores` shape (CC 3.3.2). Resolution is **DNS-free**. The reference implementation is CIRISVerify (FSD-003); every other component decodes through it.

Tied to the user, not the node (the model). The v1 **NodeCode** (`CIRIS-V1-`) renders a *node's* key and is retained as `kind: node`. The code is tied to the **entity** — five kinds mapping 1:1 onto CC 3.4.7.1 `identity_type` and the rostered `subject_kind`s:

```
kind(1): 1=user  2=agent  3=node  4=family  5=community
```

A `user` code is the owner's identity; the owner's **nodes and transport are optional**. A code that carries them resolves with no directory — first contact, a QR across a table, an air-gapped hand-off. A code that carries none resolves through the **federation directory**: the consumer walks `nodes_owned_by(U)` — the exact inverse of `owner_of` (CC 3.2; `n ∈ nodes_owned_by(U)` iff `owner_of(n) == U`, held by construction) — to find the nodes to transport to, **as visible to the walker**: a peer outside `U`'s `self` cohort holds only `U`'s federation-scoped owner-bindings (CC 5.2 audience rule), so the directory walk yields exactly the announced roster of CC 5.4.6, while `U`'s own nodes see every binding. A node whose owner resolves ambiguous is **skipped, not fatal**: one poisoned node must not make its owner unroutable, while a direct query about that node still fails closed.

Wire format (normative). Binary payload, then **CRC-16-CCITT** (polynomial `0x1021`, init `0xFFFF`, no final xor) appended as 2 bytes big-endian, then **RFC 4648 base32** (alphabet `A-Z2-7`, padding stripped on encode, re-padded on decode), prefixed by the version token and grouped into 4-character dash-separated chunks for display; the QR form is ungrouped. Decoders normalize (drop whitespace, upper-case, strip dashes) and accept the undashed prefix. All length-prefixed fields are 1-byte length, ≤ 255 UTF-8 bytes; an overflow is malformed.

```
v2 payload  (prefix CIRIS-V2-)
  version(1)=0x02 | kind(1) | sha256(key_id)(32) | ed25519_pubkey(32)
  | LP(key_id) | hint(transport) | hint(alias) | hint(group_key_id)
v3 payload  (prefix CIRIS-V3-)  -- all v2 fields byte-identical, then:
  node_count(1) = 0..=16
  node_count x [ LP(node_key_id) | transport_ed25519(32) ]
  pqc_commitment: 0x00 absent | 0x01 + 32 raw bytes = sha256(ML-DSA-65 pubkey)
```

`LP` = 1-byte length prefix + UTF-8; `hint` = `0x00` when absent, else `LP`. `group_key_id` is the family/community `*_key_id`, absent otherwise. `sha256(key_id)` binds the display `key_id` into the CRC-protected payload — a decoder MUST verify it against `key_id`. The v1 layout is retained unchanged: `version=0x01 | key_id_hash(32) | pubkey(32) | LP(key_id_str) | LP(transport_hint) | LP(alias_hint) | crc16` under `CIRIS-V1-`.

Constraints (each MUST be enforced at encoder AND decoder — a code minted by another implementation is exactly the case an encoder cannot police).

1. **An embedded node key is the node's TRANSPORT Ed25519 — never the owner's federation key, never the node's federation key.** A destination derived from a federation key is `sha256(fed)[..16]`, an explicit-hash destination that categorically cannot be announced, so no peer can self-learn a route to it. A code whose embedded transport key equals the owner's pubkey MUST be rejected.
2. **Only `kind = user` MAY embed nodes.** "The owner's nodes" is meaningless for a node, and a group's destinations are group-scoped material a code MUST NOT carry at all (CC 5.4.6). A code carries **lightnet** facts only — federation-scope identity that already announces and carries no anonymity claim.
3. **`node_count` $\leq$ 16 and total payload $\leq$ 1024 bytes before base32.** Bounding the count without bounding the size bounds the wrong thing: 16 × 255-byte ids exceeds what a QR can render, defeating the hand-off the format exists for.
4. **Empty is valid and is the default.** A code with no nodes and no commitment MUST

encode as **v2, byte-identically**, so nothing already issued moves; only a non-empty tail emits `CIRIS-V3-`.

5. **Compatibility.** v1 decodes as `kind: node`; v2 decodes unchanged; a v3 decoder accepts all three prefixes; a v2-only decoder MUST reject `CIRIS-V3-` outright rather than mis-parse it — the prefix differs before any payload byte is read.

The PQC commitment (normative). A code names an Ed25519 key and nothing else, but the substrate takes `federation_keys` writes only at `algorithm: hybrid` (Ed25519 + ML-DSA-65; CC 5.3.2.4.3.1) — so without more, a code-admitted key has no PQC half to register and first contact has no conformant path. The ML-DSA-65 public key is 1952 bytes, which would end the code's life as something a person can read aloud; the code therefore carries the **32-byte commitment** `sha256(raw ML-DSA-65 public key)`. Rules: **(a)** the host admits the classical half plus the commitment, fetches the ML-DSA body through the existing Key Pull, and verifies it against the commitment **before** writing a hybrid record — nothing is registered until both halves are present and bound, so the hybrid-only rule is preserved, not weakened; **(b)** it applies to **any** kind of code — a plain `user` code with no nodes emits v3 with `node_count = 0`; **(c)** it is presence-tagged and **last**, so a v3 code minted before it still decodes with the commitment absent; **(d)** the API form is exactly 64 **lowercase** hex characters, **rejected, never repaired** — normalizing case would let two spellings of one identity mint two codes; **(e)** a FedCode is **unsigned**: the commitment inherits exactly the trust of the code carrying it and adds no authority — a conforming implementation MUST NOT treat a commitment match as authentication of the identity; what it buys is that a Key Pull cannot be substituted after the fact. A code with **no** commitment fails closed at the pull check: there is nothing to bind the pulled key to.

`key_id` format. *(normative; FSD-003 §4)* `key_id = "<label>-<fingerprint>"`, `fingerprint` = the first **10 base32 characters** (50 bits) of `sha256(ed25519_pubkey)`, lowercased; `label` is lowercased and reduced to `[a-z0-9-]`, cosmetic. **Collision-resistant by construction, not collision-free** — the suffix is bound to the key, so two entities choosing one label collide only if their keys' hashes share 50 bits, which under one label becomes likely near $\mathbf{2^{25}}$ **keys** (the birthday bound), not 2^{50}; **verifiable** — anyone recomputes the suffix from the pubkey (a random UUID cannot do this: one could claim another's); friendly — `eric-moore-k7f3qd2pza` reads as a name. **Registry global-uniqueness is therefore load-bearing, not a backstop**: a registrar MUST refuse a second key under an already-registered `key_id`, and a consumer resolving a `key_id` MUST verify the fingerprint against the pubkey it obtains (CC 2.6.8 constraint above), so an aliasing collision is detected at registration and never at resolution. A deployment expecting more than $\mathbf{2^{16}}$ **identities under one label** SHOULD raise the fingerprint length (16 characters, 80 bits, moves the bound to 2^{40}); FSD-003 owns that change and this Part follows it.

Onboarding — usercode → owner (the chosen model). Put the owner's usercode (`kind: user`) in a node's configuration and the node becomes one of the owner's devices with **one approval tap and no PIN or QR handshake** — under the honest constraint of CC 1.13.2: owner-binding MUST be a **user-signed** `delegates_to`, so a node cannot make itself owned by reading a pubkey. On boot the node decodes the usercode, learns its intended owner `U`, and self-registers as a **pending** `identity_occurrence` **of** `U` (CC 3.3.6) — trust-and-serve only until owned; `U`'s client lists pending occurrences naming `U`; `U` approves, and their hardware-rooted key signs `delegates_to(U → node)`. The pending-until-approved window **is** the safety property: a usercode is public (a pubkey), so a stolen one lets a node *request* ownership, never *obtain* it. The rejected alternative — a usercode embedding a pre-authorized signed delegation so the node binds with no tap — is a **bearer credential** (theft = silent ownership) and is NOT the default;

if ever added it MUST be expiring, scope-limited, and revocable, by amendment.

2.6.9 Conformance language

The key words **MUST**, **MUST NOT**, **REQUIRED**, **SHALL**, **SHALL NOT**, **SHOULD**, **SHOULD NOT**, **RECOMMENDED**, **NOT RECOMMENDED**, **MAY**, and **OPTIONAL** in this document are to be interpreted as described in BCP 14 (RFC 2119 + RFC 8174) when, and only when, they appear in all capitals, as shown here.

2.6.9.1 Normative vs informative content (what interop requires)

A reader may **agree with the protocol and disagree with its philosophy** and still build a fully conforming implementation. The two are deliberately separable:

- **Normative (binding for interoperability):** the wire format and its conformance surface — the CC 2.4 structural primitives; the CC 2.1 envelope fields; the CC 3.1 namespace + `subject_kind`s; the CC 2.6.2–2.6.1 canonicalization rules; the CC 3.5 relation precedence; the CC 3.4 reserved-prefix rules; the CC 4.4 composition policies; the CC 5.3 endpoint shapes; and every RFC-2119-keyworded statement. This is exactly the surface enumerated in CC 1.7 ("report the surface beside the invariant"). **Conformance is judged against this and nothing else.**
- **Informative (explanatory framing; NOT binding):** the motivating philosophy and rationale — notably CC 1.13.1 (the Ubuntu / relational-anthropology substrate), the cross-tradition readings, and prose written as motivation rather than requirement. These explain *why* the normative choices were made; they add **no** conformance obligations. An implementer who rejects the anthropology but emits/consumes wire-correct Contributions is conforming.

Where framing produced a concrete wire consequence, that consequence is restated as a normative rule in its own section (e.g., structural invisibility is motivated informatively but enforced normatively at CC 5.2, bounded by CC 1.13.3). When in doubt, the RFC-2119 keywords and the CC 1.7 surface govern; informative prose never overrides them.

Part 5 — Transport & Substrate

5.1 Epoch keying + cascade (normative: D2 / D3)

The stream-epoch DEK seals content **O(1)**; the per-subscriber `key_grant` cascade distributes the 32-byte epoch key **O(N)/epoch** (sender-key / Megolm shape) = CC 4.4.3.4.1 Policy-L cascade applied to a community roster against a *rotating* key.

PQC at rest — `wrap_algorithm: v2` MANDATORY (normative). The epoch-DEK `key_grant` cascade MUST wrap the DEK with **`wrap_algorithm: v2`** — the hybrid X25519 + ML-KEM-768 construction (FIPS 203) — never `v1` (X25519-only). The DEK protects content that may persist indefinitely; a classical-only KEM is a harvest-now-decrypt-later exposure even though the content AEAD (AES-256-GCM, CC 5.3.3.1) and the wrap *signature* (Ed25519 + ML-DSA-65) are already PQC-safe. The versioned `KEY_GRANT_V1_INFO` HKDF context rotates cleanly to the v2 context. A Consumer MUST reject a streaming epoch grant carrying `wrap_algorithm: v1`. **Full PQC envelope for streaming**: content = AES-256-GCM (symmetric, PQC-safe) · DEK wrap = X25519+ML-KEM-768 · authenticity = Ed25519+ML-DSA-65 · hashes = SHA-256 (PQC-safe) · in-transit = CC 5.3.3.5 E1 two-layer hybrid (below).

One construction, one wire identifier (normative). The hybrid X25519 + ML-KEM-768 + AES-256-GCM + HKDF-SHA256 construction has **exactly one** wire identifier across the federation: **`x25519_mlkem768_aes256_gcm_hkdf_sha256`** (the CC 3.3.2 vocab variant `X25519MlKem768Aes256GcmHkdfSha256`). Producer, substrate, and every consumer MUST serialize this exact byte string, and MUST accept only this byte string, wherever the construction is named — the streaming epoch-DEK wrap here, the CC 4.4.3.4.1 self/family at-rest wrap, and the media-DEK wrap are the **same** construction under the **same** identifier, **not** per-field vocabularies; a consumer MUST NOT treat them as distinct algorithms and MUST NOT define a field-local spelling. The hyphen-separated form `x25519-mlkem768-aes256-gcm-hkdf-sha256` — and every other case-, separator- or ordering-variant — is a **non-conformant alias**: it MUST be rejected as an unknown `wrap_algorithm` under the CC 3.3.2 closed-set fail-secure decode rule, and a consumer MUST NOT normalize case or separators before comparison (normalization silently re-opens the ambiguity this clause closes, and would make two distinct future vocab rows collide). The snake_case form is canonical because it is the form already pinned by the CC 3.3.2 vocabulary and its `v1` sibling `hpke_rfc9180_base_x25519_aes_gcm`. An implementation currently emitting the hyphenated alias MUST migrate to the canonical string; a transitional **accept-only** allowance for the alias MAY be carried behind an explicit, default-off deployment flag, MUST NOT be emitted under any configuration, and is removed at the next MINOR.

Unlinkable presentation — required as a property, staged as a path (normative). Credential-presentation unlinkability is a **required property commitment**: it is the presentation-layer form of CC 1.13.3.4's differential-uptake rule (protection must be structural — the vulnerable cannot be asked to manage key hygiene), and an assurance rung presented across communities MUST NOT become a cross-service correlation handle. Adoption is **staged**: portability (OR-of-N) applies now; the unlinkable presentation format is a **reserved upgrade slot** with **ZK-wrapped unmodified ML-DSA as the designated candidate** (FIPS 204 untouched, no re-issuance — proofs are over signatures already made and hardware-rooted; cost lands per-relationship at the consent moment). One MUST binds the interim so the slot stays reachable: **no presentation surface may bake a mandatory stable identifier across verifiers into the wire contract** — the upgrade MUST remain a format addition, never a re-issuance event.

Identifier form is a class rule (normative). Every CC wire-vocabulary identifier — `wrap_algorithm`

and the other CC 3.3.2 closed-set strings — is lowercase-ASCII `snake_case` and is matched by **exact byte comparison**. **Domain-separation constants are a different class and are NOT subject to that rule**: `KEY_GRANT_V1_INFO` (`cewp-key-grant/v1`), the CC 5.3.3.1 STREAM-nonce label `ciris-stream-nonce/v1`, the CC 5.4.2 symbol label `ciris-edge/scope-privacy/symbol/v1`, and the CC 5.4.4 `HPKE_SUITE_ID` are literal KDF/AEAD inputs whose hyphens and slashes are load-bearing bytes; they MUST be used exactly as written and MUST NOT be snake_cased, and they MUST NOT be compared against vocabulary identifiers.

Epoch index is monotonic, per-`stream_id`, greenfield — a **separate addressing axis** from `key_grant.rotation_chain`:

Axis	Addressing	Where it lives	Supersession
Content-addressed grant supersession	`(content_sha256, recipient_key_id)`	`cirisnode_contributions` V054 partial indexes (planner-AND'd)	`rotation_chain` payload-level lineage (list of prior `key_grant_id`s); walked reader-side
Stream/epoch-addressed grant supersession	`(stream_id, epoch[, recipient])`	`federation_stream_chunks(stream_id, seq)`	Same `rotation_chain` payload-level supersession **reused on the new axis**

Epoch triggers (D3) carry the forward-secrecy guarantee — once a member leaves, subsequent content is sealed under a key they cannot derive:

Trigger	Behavior	Forward-secrecy implication
Member removal	**MANDATORY rotation** (coalesced per below; exempt for ungated public broadcasts per below) — the forward-only-unsubscribe enforcement	Subsequent epochs sealed under a DEK the removed member doesn't have
Member addition	Epoch advances — an MLS Add is a commit, so the exporter secret and every per-stream DEK derived from it change; Option-A catch-up per CC 4.5.12.1 applies only to a **kept** stream	New member derives the current epoch's DEKs from the group; prior epochs are reachable only for a kept stream, as `key_grant:stream:v1` wraps per the `history_on_join` envelope field (CC 2.1); an unkept stream has no catch-up by design
Time / bytes	Optional hygiene rotation	Operator policy; default off

Removal coalescing. At broadcast scale, naive per-removal rotation is unaffordable: at N = 10^6 and realistic audience churn (~30%/hr), one rotation per departure ≈ 83 epochs/s → a flat-unicast cascade of **~3.1 Tbps — exceeding the ~2 Tbps content fan-out itself**. The substrate therefore MUST **coalesce removals**: all removals admitted within one STH cadence window **T (default 2 s, the CC 5.3.3.6 equivocation-window grain)** are batched into a **single** epoch rotation covering the whole batch. This caps the rotation rate at 1/T **regardless of churn** (≤ 1,800 epochs/hr at T = 2 s → ~18.7 Gbps flat-unicast cascade at N = 10^6, ~0.9% of content fan-out), and bounds removed-viewer exposure to ≤ T — the same grain the stream's equivocation window already accepts. A removed member's exposure window is therefore ≤ `T`, never "until the next scheduled rotation."

Public-broadcast exemption (normative; generalised to rosters by CC 3.2 `public` open admission). A `live_stream` whose roster is **ungated** — `listed: public` AND grants issued to any requester without an admission ceremony — carries **no confidentiality claim against departed viewers** (anyone, including the departed viewer, can re-subscribe and receive the current DEK on request). For such streams, member removal MUST NOT force rotation (it buys nothing and costs the full cascade); the time/bytes hygiene rotation and the CC 5.3.3.1 nonce-space bounds still apply. Rotation-on-removal remains **MANDATORY for every gated**

roster (bounded membership, admission ceremony, or any non-public `listed` state) — there the DEK *is* the access-control boundary and the forward-only-unsubscribe guarantee is real.

Rekey conforms to MLS TreeKEM (RFC 9420, normative). The epoch-DEK rekey on member change is the MLS TreeKEM construction: an O(log N) path rekey, the commit signed once, with RFC 9420 blank-node / unmerged-leaf handling and parent-hash tree integrity. The hybrid KEM (X25519+ML-KEM-768) is the MLS ciphersuite's HPKE KEM. **One DEK per `(stream_id, epoch)`, two distributors (normative).** For the live participants of a call the group's MLS **exporter secret for that epoch is the epoch DEK** — MLS is the rekey and live distributor, and there is no second key. The same DEK is wrapped as `key_grant:stream:v1` rows (CC 3.3.2, one per recipient, `wrap_algorithm: v2`) for late joiners, recordings and anyone the audience admits — **only for a stream that is kept**: one that is recorded, or whose `history_on_join` is `full`. An unrecorded live call wraps nothing: its DEK exists only inside the live group, so an ordinary call keeps the forward secrecy MLS gives it and leaves no durable wrap that could later open it.

Per-stream DEK under the group exporter (normative). "One DEK per `(stream_id, epoch)`" is read literally: the epoch DEK of a `live_stream` is **derived** from the group's MLS exporter secret for that epoch, with an **empty exporter context** so every implementation derives the same bytes, and the stream bound by a KDF after it:

```
DEK(stream_id, epoch) = HKDF-SHA256( ikm  = MLS-Exporter(label = "ciris-stream-dek/v1", context = "",
    length = 32),
                                     salt = "",
                                     info = b"ciris-stream-dek/v1" || stream_id_utf8 || epoch_be8,
                                     len  = 32 )
```

One label serves both the exporter call and the `info` tag; `epoch_be8` is the `u64` epoch big-endian, the CC 5.3.3.1 encoding, and no length prefix is needed for the reason that section gives (fixed-length tag, fixed 8-byte suffix). A `key_grant:stream:v1` wraps **this** per-stream DEK and therefore opens exactly one stream; the exporter secret itself is never wrapped, so one participant's kept stream never opens another participant's unkept stream in the same epoch. "No second key" above means no second *distributor*; a derivation under the one MLS secret is not a second key. Conformance is proven by a known-answer test set published with CIRISVerify; the 5.3.3.1 nonce prefix is unchanged and already keys on the epoch DEK. **Kept, changed mid-stream (normative):** `history_on_join` is set per stream by its producer (CC 2.1); a call is N such streams and no room policy makes another producer's stream kept. Turning kept on mid-stream wraps the current and later epochs; earlier epochs' DEKs were never wrapped and are gone. Turning it off is forward-only: wraps already issued are not unmade (`withdraws` on a `key_grant` is refused, CC 3.3.2), and wrapping stops from the next epoch. **"Recorded" is not a field:** a stream is kept when its genesis envelope carries `history_on_join: full` or when `key_grant:stream:v1` wraps exist for it; the recording is the after-the-fact CC 3.3.13 `video` blob with `derived_from: stream_id`, and any retention promise rides the producer's `consent:deletion_sla:*`.

Delivery is decisive. The TreeKEM advantage is *multicast aggregation*, not the tree itself: with efficient multicast the commit egress is **O(log N)** (one commit serves all); over unicast the commit must reach each member, **O(N log N)** — in which case the **flat per-member cascade (O(N), one grant per member) is competitive-to-better**. The substrate MUST select per deployment based on whether the transport multicasts; the choice is **wire-invisible** (tree position rides the opaque `key_grant` payload; no schema migration). Whether the RET mesh offers efficient multicast is the open transport question (the bandwidth model's one free parameter).

Forward secrecy is forward-only by default (CC 4.5.12.1 Option A). **Post-Compromise Security (PCS)** is AVAILABLE (inherent to TreeKEM key-updates: a compromised current member heals on the next commit) and is **OPTIONAL, operator-enabled** — a deployment MAY require periodic self-updates for PCS, or stay forward-only.

Catch-up bound (P4): `min(operator depth cap [LensCore knob, NOT a substrate constant], chunk-eviction horizon)`. Three distinct windows that are NOT conflated: chunk-eviction horizon ≠ the holding claim's 72 h liveness (CC 3.1.3.3) ≠ grant durability. A catch-up request against an evicted epoch returns `ContentMiss` — **fail-honest, no silent gap** (consistent with the `MISSION.md` fail-honest invariant). Operators MUST ship the P4 cap **with** the cascade, else 10^6 grant Contributions per rekey is the unbounded worst case.

5.2 Structural invisibility: no holding claim leaves a bounded cohort

The strongest privacy guarantee in the system is the one the operator cannot switch off. The load-bearing claim from ciris.ai/cewp:

> *Self and family content never emits the attestation that would tell the rest of the network it exists. You don't need a privacy policy to keep family photos off the federation — the wire format can't carry them in the first place.*

This is codified as a normative substrate discipline. When a Contribution carries `cohort_scope: self` OR `cohort_scope: family`, the substrate MUST NOT emit any holding claim observable outside the cohort — the `custody:ack:v1` report is placed at the content's scope and replicated to members only (CC 3.1.3.3); the content's bytes are delivered to admitted members of the relevant self-collective (CC 3.3.6 `identity_occurrence`) or family (CC 3.3.4) via the at-rest encryption flow, NOT via the public holder-discovery directory.

The privacy property is structural, not policy:

- A non-member peer cannot issue `ContentFetch` for the bytes because no holding claim it can observe names a holder.
- A non-member peer cannot even *discover* the bytes exist via the substrate — the only attestations referencing them are scoped to the self-collective / family and never federate beyond it.
- This is the wire-format-level closure of the cewp **structural invisibility** claim: privacy emerges from format constraints, not from operator policy or legal undertaking.

> ***Scope (normative):*** *structural invisibility buys* ***content-holding confidentiality only*** *— it is NOT relationship-existence, metadata, traffic-analysis, or unobservability privacy. The bounding non-goals are stated canonically at CC 1.13.3.1; do not represent CEG as providing the stronger properties.*

Substrate enforcement:

```
On admission of a Contribution C with cohort_scope in {self, family}:
    # (1) STRUCTURAL INVISIBILITY -- UNCONDITIONAL (the cewp privacy promise):
    substrate MUST NOT emit a holding claim for C's evidence_refs bytes outside the cohort
    substrate MUST NOT propagate C beyond the self-collective / family scope
        via any other directory or discovery surface
    # (2) AT-REST ENCRYPTION -- the S8.1.12.4 cascade (defense-in-depth):
```

```
WHEN self/family at-rest encryption is enabled for the deployment:
    recipients := if cohort_scope == self:   all current identity_occurrences of C.attesting_key_id
                  if cohort_scope == family: all current members of family per C.family_id
    FOR each recipient r:
        kem := resolve_encryption_keys(r.key_id)    # current occurrence's encryption_pubkeys (S5
 .6.8.8.2)
        IF kem is None OR kem.ml_kem_768 invalid:
            # FAIL-SECURE: no valid v2 wrap target ? EXCLUDE r from the grant.
            # MUST NOT fall back to plaintext or to wrap_algorithm v1.
            # NOT SILENT: emit hard_case:recipient_excluded:{scope_key_id}
            #   (S7.7, which defines the closed reason-set) INTO the self/family
            #   scope -- recipient r, reason, + the skipped Contribution's envelope
            #   ref. Scoped to the cohort; never federates beyond it (S10.1.4
            #   invisibility preserved).
            skip r   # content stays encrypted + unreachable to r until r registers encryption_pubkeys
        ELSE:
            substrate MUST wrap C's DEK via key_grant (S5.6.8.4, wrap_algorithm v2 -- S8.1.12.4)
                to kem.{x25519, ml_kem_768}
```

Two layers, not one. (1) **Structural invisibility** — holding claims that never leave the cohort + non-propagation beyond scope — is the **unconditional** privacy promise (the cewp "the wire format can't carry them" claim); it holds even when the at-rest bytes are plaintext, because no discovery attestation federates. (2) **At-rest encryption** — the CC 4.4.3.4.1 DEK cascade — is **defense-in-depth** (against local-disk forensics, the host operator, or the cloud-substrate operator); it is operator-policy and MAY default off as a v1 migration posture, **but when enabled MUST use `wrap_algorithm: v2` (hybrid PQC, CC 4.4.3.4.1)** — never v1. The CEG-RET-native target is at-rest-on for self/family (the "everything PQC at rest" standard).

Composition with at-rest encryption flow: when self/family at-rest encryption is enabled, persist wraps the DEK (`wrap_algorithm: v2`) under each currently-admitted `identity_occurrence`'s `occurrence_key_id` (self) or each `member.key_id` in the named family's roster (family). New occurrence / new family-member admission triggers retroactive `key_grant` emission for all extant `cohort_scope: self|family` content (the "I bought a new phone and want my Twitter history" / "I added Carol to the household" flows from CC 3.3.4 worked example).

Recipient encryption-key resolution + fail-secure exclusion. The wrap target is **not** a recipient's signing key. `wrap_algorithm: v2` needs the recipient's `{x25519, ml_kem_768}` **content-KEM** keys, which the recipient self-certifies via its `identity_occurrence.encryption_pubkeys` (CC 3.3.6.1); the substrate resolves them by `resolve_encryption_keys(key_id)` = the recipient's current (non-superseded, within-`valid_until`) occurrence → its `encryption_pubkeys`. Because this layer **mandates v2**, a recipient whose current occurrence carries **no valid ML-KEM-768 key MUST be fail-secure *excluded*** from the grant — the content remains encrypted and unreachable to it; the substrate MUST NOT fall back to plaintext or to `wrap_algorithm: v1`. To be an at-rest-encryption recipient, an identity MUST have a federation-present occurrence carrying `encryption_pubkeys`. This is the non-maleficence / fail-secure default: a missing key denies access, never downgrades the protection.

Exclusion MUST NOT be silent. A bare `skip` makes a fail-secure exclusion indistinguishable from "the family went quiet" — a soft-censorship vector for a buggy or malicious substrate, and a contradiction of the spec's own `hard_case:*` attestability grain. On every fail-secure skip the substrate MUST emit **`hard_case:recipient_excluded:{scope_key_id}`** (CC 3.4.4, which defines the closed `reason` set) **into the affected self/family scope itself** — carrying the excluded recipient's `key_id`, a `reason`, and the skipped Contribution's envelope ref — so the excluded member (who still sees cohort-scoped attestations) has something to audit and remediate. The event is cohort-scoped: it MUST NOT federate beyond the self/family (the CC 5.2 invisibility promise is preserved; the *fact* of the family's content is not leaked by its exclusion

events).

Locality dividend (cewp claim): the structural invisibility mechanism is *why* ~65% of activity stays local in the cewp scaling model — `cohort_scope: self|family` content is the bulk of daily activity (family photos, personal notes, in-household device chatter), and that bulk never federates. Operators do not configure this; the wire format enforces it.

Boundary cases:

- One rule at every tier (CC 6.1.5.3): `cohort_scope: community | affiliations` content's holding claims reach the community's members only; `federation` (Commons) content's holding claims are at federation scope and are the holder directory. Self/family is not a special case but the audience rule applied to the smallest roster. Within every cohort, holding claims, custody reports, target replication and anti-entropy run exactly as at every other tier.
- A `cohort_scope: self` Contribution that is later promoted via `supersedes` to `cohort_scope: community` emits its holding claims at the new scope, on the NEW Contribution, at promotion time. The original `cohort_scope: self` Contribution's bytes remain structurally-invisible at federation; only the promoted scope's bytes propagate.
- `cohort_scope: self` content with `subject_key_ids` containing a non-self party (e.g., a private note Alice writes ABOUT Bob) is admitted and stays in Alice's self-collective; Bob does NOT receive a key_grant unless Bob is also in Alice's self (not the case for two distinct identities). Bob's CC 2.3 subject-side revocation authority over the note still composes, but the bytes never reach Bob without Alice's explicit re-emit at a higher cohort_scope including Bob.

5.3 Endpoint shapes

CEG specifies five public + one admin HTTP endpoint shape for the discovery + cosigning surfaces. Wire-format consumers read these.

5.3.1 STH cosigning + witness directory

The witness directory is where transparency becomes accountable: independent witnesses cosign the log's tree heads so a producer cannot show one history to one party and a different one to another. Consumers verify cosignatures and the witness quorum; CEG's emission half:

`POST /v1/transparency/sth/cosign` (public)

Witness posts cosignature on `(tree_size, root_hash, signed_at)`. Registry verifies hybrid Ed25519 + ML-DSA-65 against witness pubkey in directory; persists on success.

Request body:

```
{
  "tree_size": <int>,
  "root_hash_sha256_hex": "<64-char-lowercase>",     // per S0.6
  "signed_at": "<rfc3339_canonical>",                // per S0.5
  "witness_key_id": "<string>",
  "ed25519_signature_b64": "<base64-url>",
  "mldsa65_signature_b64": "<base64-url>",
  "consistency_proof_root_hash_sha256_hex": "<64-char-lowercase>",
```

```
  "consistency_proof_tree_size": <int>,
  "consistency_proof_path_b64": ["<base64-url>", ...]
}
```

Canonical bytes (witness MUST sign these):

```
canonical = sha256(
    "ciris.sth_cosign.v1\n" ||
    "tree_size=" || decimal_no_leading_zeros || "\n" ||
    "root_hash_sha256=" || sha256_hex_lowercase || "\n" ||  // per S0.6
    "signed_at=" || rfc3339_canonical                      // per S0.5
)
```

Ed25519 over `canonical`; ML-DSA-65 over `canonical || ed25519_sig` (bound payload).

`POST /v1/transparency/lineage-head/cosign` (public) — trust-root lineage heads

The same directory witnesses **trust-root lineage heads** (CC 3.2 T6): the lineage's signed roster record at a version, which T6 makes the head — named by the SHA-256 of its signed bytes. A witness posts its cosignature; the Registry verifies hybrid Ed25519 + ML-DSA-65 against the witness's key record in the **charter's** witness directory for that lineage (never a directory of its own), refuses a witness that is not independent of the lineage under T6 at the cosignature's `signed_at`, rejects a `signed_at` more than 5 minutes from its own clock at receipt (the CC 2.6.7 bound the STH route already applies, so a witness can neither pre-date nor post-date a re-commitment), and persists on success.

Request body:

```
{
  "lineage_key_id": "<community_key_id or family_key_id>",
  "head_digest_sha256_hex": "<64-char-lowercase over the signed roster record at that version>",
  "prior_head_digest_sha256_hex": "<64-char-lowercase -- the head this witness last cosigned for this
     lineage; empty on its first>",
  "signed_at": "<rfc3339_canonical>",
  "witness_key_id": "<string>",
  "ed25519_signature_b64": "<base64-url>",
  "mldsa65_signature_b64": "<base64-url>"
}
```

Canonical bytes (witness MUST sign these; the descent claim is inside the preimage, so a relayed cosignature proves which link it attests):

```
canonical = sha256(
    "ciris.lineage_head_cosign.v1\n" ||
    "lineage_key_id=" || lineage_key_id || "\n" ||
    "head_digest_sha256=" || sha256_hex_lowercase || "\n" ||
    "prior_head_digest_sha256=" || sha256_hex_lowercase_or_empty || "\n" ||
    "signed_at=" || rfc3339_canonical
)
```

Ed25519 over `canonical`; ML-DSA-65 over `canonical || ed25519_sig`. The domain label is distinct from `ciris.sth_cosign.v1`, and the byte layouts differ, so a head cosignature can never be replayed as a tree-head cosignature, nor a cosignature as a head, nor the reverse.

Descent, re-commitment, and the two evidence tiers (normative). A witness MUST verify, before signing, that the head **descends from, or equals,** the prior head it cosigned for that lineage (equality is the T6 cadence re-commitment — the witnesses' duty, not the founders' — and MUST be accepted; descent is the `prev_head_digest` chain, each intermediate head's quorum checked against its parent's fold and its cosignatures against its parent's directory; on a witness's first cosignature the prior is the lineage's genesis head from the bundle). The Registry

MUST reject a cosignature that names a prior it did not record for that witness, or a head that neither equals nor descends from it → `LINEAGE_HEAD_INCONSISTENT` (CC 5.3.6.1); that is the malformed case. Two other cases are evidence, not errors. A head carrying a signature valid under the lineage's quorum at that head's instant that does not descend from the lineage's *witnessed* head is a **fork attempt**: the Registry MUST record it once per `(lineage_key_id, fork point)` — the nearest ancestor on the witnessed chain — MUST NOT cosign it, counts and does not store further attempts off the same point, and raises `hard_case:lineage_fork_attempt` (CC 3.4.2) — it never freezes anything. Two heads that each reach the charter's `witness_quorum` are **equivocation**, `hard_case:lineage_equivocation`, and freeze head adoption under T6; because the quorum is a strict majority of the directory, at least one witness signed both, and the two cosignatures name it. A lineage whose newest counted cosignature is older than its charter's `witness_cadence_secs` is *silent*: reported on the trust surface as a liveness signal under CC 3.2 T4, never a validity leg — its effect is on *attaching* (T4a), where a silent lineage becomes unattachable once the window lapses and attachable again on the next re-commitment.

5.3.1.1 Consistency-proof requirement (normative)

A cosignature only means something if the witness has checked that the new tree is a consistent extension of the one it last saw — otherwise quorum is agreement on a string, not on a history. A witness signing an STH MUST first verify a consistency proof from the prior STH it cosigned (or from genesis if it is the witness's first cosignature against this log). The Registry MUST reject `POST /v1/transparency/sth/cosign` requests that omit the `consistency_proof_*` fields OR whose consistency proof does not verify against the named prior STH. `witness_quorum_met` is therefore "quorum on log consistency," not "quorum on a string."

The cosign request carries `consistency_proof_path_b64[]` (base64 RFC 6962 §2.1.2 node hashes). The Registry anchors the check against the prior `(tree_size, root_hash)` **it recorded** for that witness — not a root the requester claims — and rejects: missing proof when a prior STH exists or a tree_size behind the witness's prior cosigned STH → `MALFORMED_REQUEST`; a proof that does not reconstruct both roots → `CONSISTENCY_PROOF_INVALID` (CC 5.3.6.1). A witness's first cosignature is exempt ("from genesis").

5.3.2 Transport substrate for byte-level content

Wire-format Attestations carry claims; they don't carry bytes. When a claim's `evidence_refs[]` cites a SHA-256-addressed blob (e.g., an installer binary, a config file, an adapter package per `agent_files:*` per CC 3.1.7 / CC 3.1.10), the bytes travel via Edge transport substrate: `MessageType::ContentFetch` + `ContentBody` + `ContentMiss`. Holder-discovery via the `custody:ack:v1 here` rows at the content's scope (CC 3.1.3.3); peer-resolution via Edge's `PeerResolver::resolve_holders`. NodeCore node-mode peers serve the bytes per their MISSION CC 2.4 cohabitation contract. Attestation envelope shape unchanged; SHA-256 in `evidence_refs[]` becomes universally resolvable to bytes through the substrate.

5.3.2.1 Holder directory TTL + ContentMiss feedback

A `custody:ack:v1 here` report is live for **72 hours** from its instant (CC 3.1.3.3); after that the holder reads *unknown*. Consumer policy MUST attempt at most 2 holders in parallel and accept the first successful full-SHA verification. On `ContentMiss` (holder no longer has the blob) the

consumer MUST NOT withdraw the holder's report — only its signer retracts a holding claim (CC 3.1.3.3 rule 4) — it records the miss and downweights the holder in `PeerResolver::resolve_holders`; the holder's own next acknowledgement reports `none`.

5.3.2.2 Consent revocations are NOT local-tier-eligible

A user's withdrawal of consent is the one signal federation peers rely on to stop propagating that user's data — so it can never be left unsigned and invisible to them, even briefly. The CEG-native agent design proposes **local-tier signature deferral** — self-attestations skip the hybrid Ed25519 + ML-DSA-65 signature path locally and only sign at federation-tier promotion. This is sound for **producer-only-authority self-attestations**. **The discriminator is *who holds revocation authority*, NOT whether `subject_key_ids` is empty**: a producer-authority self-attestation MAY name a subject in `subject_key_ids` per CC 2.3.1 — e.g. `observed:user:{hash}:*`, `epistemic:about:{key}:*`, a self-`consent:partnered:{user}`, or the CC 2.3.4 self-`identity:current` with `subject_key_ids=[self]` — and remains local-tier-eligible, because the producer holds the authority and no *other* subject can revoke it. The single carve-out is below: a Contribution where a **subject other than the producer holds revocation authority** over it.

Consent revocations from subjects MUST NOT use the local-tier deferral path. When a Contribution carries non-empty `subject_key_ids`, any subsequent `consent:state:revoked` emission OR `withdraws` admitted under CC 2.4.1.1 rule 2 or 3 from a subject in that set MUST promote to federation-tier within a bounded window. Default window: **24 hours** (operator-tunable per local policy).

Rationale: subject-side revocation is the wire-format observability primitive that federation peers depend on to honor consent. If a user revokes consent in the local-tier scope of one agent's substrate, and that revocation is unsigned + unpromoted for an extended window, other federation peers continue propagating the user's data — exactly the failure mode this closes.

Substrate emission: substrate MUST emit `hard_case:consent_revocation_promotion_overdue` when a subject-side revocation has been local-tier for longer than the operator-configured window without federation-tier promotion. LensCore composes `detection:consent:promotion_delay_pattern` on top (operator monitoring; not a slashing trigger on its own).

Composition with the self-attestation pattern: the CEG-native agent's `consent:partnered:{user_key}` self-attestation (producer-side stance) MAY ride local-tier; the user's `consent:state:revoked` against the agent's stance Contribution (subject-side) MUST NOT. This preserves the cardinality win of deferral while closing the leak window.

AV-61 consent-tail: transit, not rest. The "no local subject-revocations" rule and the consent-SLA promotion gate above are the **same invariant viewed from the two gates**, not a contradiction. A subject-side consent revocation — a subject-emitted `consent:state:revoked`, or a `withdraws` admitted under CC 2.4.1.1 rule 2/3 from a subject named in the target's `subject_key_ids` — is **federation-tier by classification**. It MAY *transit* the local-tier write path while in flight (the producing occurrence's substrate accepts the unsigned envelope before promotion), but it MUST NOT *rest* there: a subject revocation is **never a durable local row**. Its only conformant terminal states are *promoted* (federation-tier, hybrid-signed per CC 5.3.2.4.3) or *overdue-flagged* — never *settled local*. The substrate MUST drive it to federation-tier promotion within the 24-hour SLA defined above; the local tier is a momentary staging buffer for it, not a resting place.

The overdue emission IS the SLA enforcement. When the 24-hour window lapses without

federation promotion, the substrate MUST emit `hard_case:consent_revocation_promotion_overdue` (as required above). This emission is not merely observability — it is the signal the consent-promotion path keys off: the consent-promotion gate and the CC 5.3.2.4.3 `local` ⟹ `caller is the producing occurrence` read-gate read the **same** overdue state. Because both gates key off one shared signal, a subject revocation can never be simultaneously (a) invisible to federation peers under the local read-gate and (b) un-flagged by the consent-promotion gate. Keeping the two gates reading one state is exactly what closes **AV-61** (the two gates de-synced, CC 5.3.2.4.3): there is no window in which a subject revocation persists as an authoritative-looking local row while federation peers continue propagating the subject's data.

5.3.2.3 Cross-region merge intents: CEG-declared per subject_kind (normative)

With operational data joining the CEG-native replication stream (CC 3.3.9), the substrate runs **more than one merge policy**. The policy is a **normative property of the subject_kind, declared here** — the substrate reads and dispatches on the declaration; it MUST NOT infer policy per record type (the substrate enforces declared merges; it does not invent policy).

subject_kind(s)	Merge intent	Semantics
`organization`, `org_membership`	`lww_skew_bounded` + `withdrawal_forward_only`	Stable-id grouping (CC 3.3.9); an admitted `withdraws` (deactivation) is forward-only — a later non-withdrawn write does NOT resurrect; else latest `asserted_at` wins; tie-break smallest `attestation_id` (CC 3.5.1). The forward-only here is the **lightweight authz flag**, NOT the CC 4.4.3.4 DEK-cascade crypto path — Commons tier, no DEK exists.
`partner_record`	`monotonic_quorum` (the CIRISPersist V058 R1/Q1 machinery, generalized from `revoked_key_id` to `license_id`)	Admission anti-rollback first: a write whose `revision` decreases never enters the merge. Then the `MergeBallot` comparator: `quorum_weight` → signed timestamp → content hash. Quorum-above-time is deliberate — it neutralizes timestamp front-running (F-AV-FRONTRUN) on the records where it matters most. More-restrictive state wins on conflict: `revoked` > `suspended` > `active`.
`revocation` + the three membership-revocations	V058 R1/Q1 (the original `monotonic_quorum` instance)	As shipped.
keys / attestations / occurrences / families / communities / location_proofs	Content-addressed idempotent admission	Same content → same `envelope_hash` → dedup; rotation collisions rejected non-destructively.

Skew-bounded admission. *(normative; the LWW front-running fix)* For every subject_kind merging by `lww_skew_bounded`, the substrate MUST reject at admission any envelope with `asserted_at > now + tolerance`, where `tolerance` is the CC 2.6.7 clock-skew bound (±5 minutes; the existing `CLOCK_SKEW_VIOLATION` error class). Without this, a signer with a forward-skewed clock future-dates `asserted_at` and wins LWW indefinitely; with it, a forward-dated write can win for at most the skew window. This matters precisely because `org_membership` carries `role: OrgAdmin` — unbounded LWW on authz data is a role-escalation surface.

The two quorums (restated from CC 3.3.9 — different layers, different stewards):

the **steward-signature admission quorum** (M-of-N signature set over identical JCS bytes; the CC 3.2 founder-quorum machinery at admit — Verify's layer) is NOT the **region merge quorum** (`quorum_weight`, the `MergeBallot` tier-1 ordering above — the substrate's layer). The substrate's merge logic never counts steward signatures; Verify's admission check never orders merges.

5.3.2.4 The attestation tier model: local-tier write, query, promotion (normative)

Pins the **shared attestation surface** the four implementations (CIRISAgent, CIRISNodeCore, CIRISLensCore, CIRISRegistry) write/query/promote through, so it is analyzable + modelable as one contract. The substrate exposes the *methods*; CEG pins the *wire/conformance semantics* every implementation MUST agree on. **No 1+4 change** — a "tier" is recorded state on an attestation, not a new primitive. The tier split serves the same fail-secure cost discipline as streaming: cheap, unsigned writes stay private and local until the costly hybrid signature is paid exactly once, at the moment authority actually crosses to the federation.

5.3.2.4.1 Local-tier eligibility: the discriminator is *revocation authority*, not subject-set emptiness

A write is local-tier-eligible iff **the producer holds sole revocation authority** over it. The discriminator (*revocation authority*, NOT empty `subject_key_ids`), the producer-authority-with-named-subject examples, and the single carve-out (a Contribution where a subject other than the producer holds revocation authority — a `withdraws` under CC 2.4.1.1 rule 2/3 or a subject-emitted `consent:state:revoked`, which MUST go signed / promote per the CC 5.3.2.2 24-hour obligation, never local) are defined canonically at CC 5.3.2.2. Tier-specific addition: `witness_relation` MUST be `self` for any local-tier write.

5.3.2.4.2 Promotion: `local` → `federation` (the deferred-signature moment)

Promotion computes the hybrid signature and flips the row federation-visible. It is **idempotent** (promoting a `federation` row returns it unchanged), and at the promotion instant the tiered-scope promotion (CC 4.4.3.3.1) and the holding claim at the new scope (CC 3.1.3.3) fire exactly as for any federation write — **promotion *is* the federation-emit moment**.

Canonical bytes — `JCS(envelope)` per CC 2.6.1 / RFC 8785:

- The signature MUST cover `JCS(contribution_envelope)` — the **identical** canonical bytes any natively-federation attestation signs. A promoted row is therefore **byte-indistinguishable on the wire from one born federation-tier**; there is no "was-promoted" marker in the signed bytes.
- **Substrate columns are NOT in the canonical bytes.** `tier`, `promoted_at`, and any other storage bookkeeping are substrate state, never part of `JCS(envelope)`. Only the CC 2.1 envelope member set is canonicalized.
- **CC 2.6.1 omit-vs-materialize is load-bearing here.** Promotion MUST canonicalize the **exact member set the producer committed at local-write time** — a field *omitted* at local write MUST NOT be materialized at promote, and vice-versa, or the recomputed

bytes diverge from what a peer verifies and the hybrid sig fails. The substrate serializes the stored row → the committed envelope → JCS → sign; it MUST NOT re-default.

- Registry owns the exact envelope member set (CC 2.1 + the CC 2.6.1.2 catalog); Verify recomputes the identical JCS bytes. Do NOT use Verify's internal length-prefixed `signing_bytes` framing — that is for verify-to-verify primitives that never cross the four-impl boundary as JSON; a promoted attestation is the opposite.

5.3.2.4.3 The two tiers

Tier	Signature	Federation-visible	Read-visibility	Written by
`local`	MAY be absent (deferred per CC 5.3.2.2)	**No**	**only the producing occurrence** (self-read); every other caller — even an authorized family/community peer — sees nothing	local-tier write
`federation`	hybrid Ed25519 + ML-DSA-65 **present**	Yes	per CC 2.1 `cohort_scope` + the CC 5.2 invisibility rule	direct signed write OR `local` → `federation` promotion

Invariant (substrate MUST enforce): `tier = federation` $\Longrightarrow$ `hybrid signature present`. Nothing crosses to federation-visible unsigned. A `local` row is **labelled** local and MUST NOT be served as federation-authoritative (fail-honest). The read-gate is **orthogonal** to `cohort_scope`: it is an additional filter (`local` $\Longrightarrow$ `caller is the producing occurrence`), composing with the CC 5.2 target-membership predicate. Threat entries: AV-59 (local row leaked to a non-self caller), AV-60 (unsigned local served as authoritative), AV-61 (the two gates de-synced).

5.3.2.4.3.1 The PQC half is MANDATORY at admission: no classical-only, no hybrid-pending accommodation (normative)

The CC 5.3.2.4.3 invariant `tier = federation` $\Longrightarrow$ `hybrid signature present` is **enforced at the admission gate**, and "present" means **verified**: every federation-tier admission gate MUST check **both** halves — Ed25519 over `JCS(envelope)` **and** ML-DSA-65 over `JCS(envelope)` $\|$ `ed25519_sig` (the CC 3.1.2.1 bound-payload form) — and MUST **reject** a federation-tier Contribution that carries only the classical half. This binds **all operational-authority admission gates**, explicitly including `operational_admit` (key_grant / partner / org-membership / license writes), the CC 3.3.6.2 `transport_destination` binding, and the CC 3.2 `partner_record` / founder-quorum gates.

This is an immediate requirement — not a phased cutover. There is no opportunistic-acceptance window and no fleet-floor or calendar trigger. The rationale is exactly the threat hybrid exists to defend: an adversary holding a future Ed25519 break could forge a grant / binding / partner-record and have it admitted if the classical half alone sufficed. There is **no `require_hybrid: false` posture** — a verifier's hybrid-required check is always-on, never an operator toggle; the only conformant state for a federation-tier key is "carries a valid ML-DSA-65 half." A key that has not completed PQC wiring is **not conformant for federation-tier emission** and is confined to local-tier (self-read, CC 5.3.2.4.1) until it does. The mandate is at the **federation admission boundary only** — the single place authority crosses; local-tier rows MAY still defer the signature per CC 5.3.2.2.

The mandate binds the durable store + replication path, not only the authority gates. "Federation-tier" means **every** federation-tier attestation, including the bulk **per-trace / store-and-replicate** path — not merely the operational-authority gates enumerated above. A federation-tier trace written to the durable, content-addressed, replicated corpus MUST carry (and a verifier MUST check on ingest) the ML-DSA-65 half exactly as a `key_grant` does; there is **no "operational authority vs testimony leaf" exemption** (federation root and the trace leaf are bound by the same rule). **Content-addressing is NOT a defense against forge-later:** a CRQC-era adversary who breaks Ed25519 mints a backdated trace under a historical key and hashes **their own** forgery — the content hash matches by construction, so the address proves nothing about authenticity. Because the trace store is *kept for posterity* (it outlives the classical primitive), the per-trace signature is the single most forge-exposed surface in the federation — the "store at massive scale" CEWP crux — and the PQC half is mandatory there for the same forge-now / harvest-now-exploit-later reason as the at-rest DEK cascade (CC 5.1 / CC 4.4.3.4.1). A substrate persisting/replicating a federation-tier row whose envelope signature lacks a valid ML-DSA-65 half MUST reject it at the ingest gate — store-then-quarantine is non-conformant.

5.3.2.4.4 Query: open-prefix dimensions, bounded operators (normative)

The uniform read filters on `(dimensions[], valid_at, confidence_floor, subject_key_id?, scope)`.

- `dimensions[]` **is an OPEN-vocabulary set of prefix strings, matched by hierarchical prefix — NOT a closed enum.** Per CC 4.5.1.1 axis-vocabulary discipline, dimension prefixes are open vocab (new families — `consent:*`, `detection:community:*`, `settlement:*` — ship continuously). A closed enum would force a substrate redeploy per CEG namespace addition and break forward-compat. A query for `detection:*` matches `detection:correlated_action:bribery` etc.; an exact string matches exactly.
- **The bounded surface is the *operator set*, not the *vocabulary*. **The apophatic discipline (a fixed, named surface — not an OLAP/graph engine) is preserved by the** closed set of five predicates** above + no caller-composed SQL/projections — NOT by closing the dimension vocabulary. *Open data, closed operators.*
- `valid_at` filters `asserted_at` $\leq$ `valid_at < COALESCE(expires_at,` ∞`)`; `confidence_floor` filters `weight` $\geq$ `floor`; `scope` applies BOTH the CC 5.2 target-membership gate AND the CC 5.3.2.4.3 tier gate. Write-side reserved-prefix emitter gates (CC 3.4.7.1) are unaffected — read is gated by scope, not by dimension authority.

5.3.2.4.5 For holistic analysis + modeling (informative)

The tier model is the *same KEM-then-symmetric placement* the streaming model uses (CC 5.3.3), applied to attestations: the **expensive op (hybrid sign) is on the cold path** (promotion, at federation-emit), while the **hot path (local self-attestation write) is O(1) and unsigned**. Cost shape for a node:

- **local write**: O(1), no asymmetric crypto — the agent's steady-state memory/config/consent/identity churn.

- **promotion**: one hybrid Ed25519+ML-DSA-65 sign (~the CC 5.3.3-model per-op cost; ML-DSA-65 sign ~330 μs) + JCS canonicalization — only at the federation-emit moment, never per local write.
- **query**: the CC 2.1/DAS scope-filtered read; cost is the predicate + index, independent of tier.
- **observability for modeling**: `local` row count, promotion rate, and `hard_case:consent_revocation_promotion_overdue` count are the three measurable signals; the CC 5.3.2.2 24-hour window is the one tunable. A model of a federation's attestation load is therefore *(local-write rate, promotion rate, query rate)* with promotion carrying the only asymmetric-crypto cost — the dual of the streaming model's epoch-rekey tail.

5.3.2.5 Full-SHA verification before consumption (normative)

A CEG-Conforming Consumer (CCC) MUST verify the full SHA-256 of received bytes against the value in `evidence_refs[]` BEFORE handing the bytes to any consumer (Agent loader, Portal renderer, etc.). A holding claim names the blob by its address digest (CC 3.1.3.3); the consumer MUST verify the full SHA of the received bytes regardless of how it found the holder. Bytes that fail the full-SHA check MUST be discarded and the holder MUST be reported per consumer policy (a negative score on the holder; never a `withdraws` of another device's holding claim).

Every blob carries its size, and size is checked first (normative). A descriptor that cites a blob from `evidence_refs[]` — installer, config, adapter package, media, any bytes — carries the blob's byte length alongside its digest. A producer always knows it at creation time. **Where the size lives:** an `evidence_refs[]` entry is a bare URI or content-hash string on the frozen CC 2.1 envelope and carries no size; the size rides the **descriptor** whose canonical shape this document already defines — the multimedia Source struct's `{digest, size}` (CC 3.3.13), the live-stream chunk-descriptor list (same section), the per-artifact entries of `SkillImportManifest` and the build manifest, and the `build` descriptor of a build-manifest Contribution, which carries `manifest_size` beside `manifest_hash` (CC 3.1.2.1). This rule binds every blob cited **through** such a descriptor. A blob cited only by a bare `evidence_refs[]` string has no declared size, and a consumer fetching it enforces the CC 2.6.1.3 cap instead — it MUST NOT invent a size field on the envelope, which would be a wire change; a producer that needs the size check on such bytes cites them through a descriptor. The consumer MUST compare the received length to the declared size **before** computing the digest: a mismatch is a refusal without reading the rest, and the descriptor alone lets a receiver enforce caps and refuse a `ContentFetch` that would overrun. Size is not a substitute for the digest, only the cheaper check that runs first. The one descriptor with no total size is a **live stream in progress** (CC 3.3.13 `live_stream`): each chunk and the init segment carry their own, and the recording, once the stream ends, is an ordinary blob with one.

5.3.2.6 The render tier is receiver policy, computed from verified, sniffed bytes (normative)

A CCC MUST decide whether and how to render a blob from the **verified bytes and its own policy, never from a claim in the descriptor**. After CC 5.3.2.5 verification the consumer MUST sniff the leading bytes and require the sniffed essence to **equal** the declared `format` (CC 3.3.13); a mismatch is a **refusal, not a correction**. Where the description is sealed (`sealed_descriptor`, CC 3.3.13), the sniff runs where the bytes decrypt, by the party that opened them,

against the `format` inside the seal; a holder that cannot open the bytes does not sniff and does not claim to. The declared `format` / `codec` are inputs to that comparison and to capability checks, never authorisations. This is exactly how Element ignores `info.mimetype` for the blob-URL decision and Synapse ignores the uploader's wishes for `Content-Disposition`; it is what makes the set below a property of the *receiver* that a malicious sender cannot widen.

Recommended renderable set (recommended CCC policy — not normative). The intersection of what Signal, WhatsApp, Mastodon, Threema and Element *emit*, what Android / iOS / Skiko / Chromium all decode, and what has a memory-safe decoder today. A CCC MAY adopt a narrower set; a wider one is its own risk to own, and this Part does not endorse it.

`format` (sniffed)	Tier	Conditions
`image/jpeg`, `image/png` (APNG as PNG), `image/gif`	**A — render**	≤ 16 MB, ≤ 33 Mpx; GIF ≤ 921,600 px + frame cap; reject bytes after `IEND` / `FFD9`
`image/webp`	**A — conditional**	≤ 10 MB; node runs Signal's `webpsan`; re-encoded on ingest
`text/plain`	**A**	valid UTF-8, ≤ 1 MB, bidi controls rendered visibly
`video/mp4`	**A — conditional**	`codec` present and `avc1.*` (≤ L4.1) + `mp4a.40.2` only; ≤ 100 MB, ≤ 8,294,400 px; node runs `mp4san`; receiver rejects unlisted tracks
`audio/mp4` (`mp4a.40.2`), `audio/mpeg`	**A**	ID3v2 ignored — never decode `APIC`
AVIF, HEIC / HEIF, JXL, WebM, MOV, MKV, Ogg / Opus, FLAC, WAV, ADTS, Markdown	**B — sender converts to a Tier-A form before hashing; raw arrival = C**	iOS has no WebM / Opus playback; Skiko decodes none of the images; Mastodon disabled HEIF 2026-09-15
`image/svg+xml`	**C for the bytes — never a client renderer or WebView.** A node MAY emit a PNG rendition (`derived_from`) from a hardened rasteriser (`resvg`: no file-href resolver, no system fonts, caps) in an isolated process; that PNG is Tier A	excluded by name in Signal, Session, Threema, Element, Synapse, Delta, Mastodon. The CC 3.3.13 rendition rule dissolves the B-or-C question: the SVG is C, the node's PNG is A
`application/pdf`	**C — download**; node MAY rasterise pages to attested PNG in an isolated process	JS / Launch actions / embedded fonts; pdf.js CVE-2024-4367, PDFium CVE-2024-7973 / 5846
`model_3d` (glTF, USDZ, FBX, splats)	**C — download**; node MAY rasterise a PNG poster in an isolated process	33 Apple USD CVEs through CVE-2026-20616; 15 Pixar OpenUSD; FBX SDK CVE-2026-10709 / 10710; glTF: cgltf CVE-2026-32845, VTK CVE-2025-57108, Gitea CVE-2026-28737 — stored XSS *through a glTF field* in a 3D viewer
HTML, Office, fonts, archives, TIFF, PSD, BMP, ICO, everything else	**C — refuse**	generic card, external handoff, dangerous-extension block on save

Where enforcement lives. The ingest pipeline that makes Tier A enforceable — size (CC 5.3.2.5: checked first) → full-SHA → sniff (only on verified bytes, CC 5.3.2.6) → memory-safe demux / decode → canonical re-encode → strip metadata → emit rendition + descriptor — belongs

on the **node**, so every client renders only bytes the node's safe encoder produced. Memory-safe decoders are production today: Chromium's default PNG decoder is the Rust `png` crate, JPEG is moving to `zune-jpeg`, Signal ships `mp4san` / `webpsan`, Android 17 moves AAC in-process as Rust. Images, SVG and audio decode entirely in `#![forbid(unsafe_code)]` Rust; video is "Rust validates container and elementary stream, hardware decodes".

5.3.3 Streaming transport, per-stream logs & delivery receipts

This is the **delivery axis** — the third orthogonal envelope concern alongside visibility (`cohort_scope`) and revocability. CC 2.4's 1+4 primitive set is untouched; this is an endpoint + envelope + composition extension, NOT a grammar change.

Bifurcation:

Half	Cardinality
Observer-share / directed delivery (single Contribution → subscriber-set; no `stream_id`; rides CC 3.3.2 `key_grant` + CC 4. 4.3.2 Policy M membership)	N=1 typical; per-subscriber `key_grant`
Media / streaming multicast (`live_stream` chunk-DAG; per-`(stream_id, epoch)` keys; specified by CC 5.3.3.1 / CC 5.1 / CC 5.3.3.6)	N>1; flat per-epoch `key_grant` cascade

5.3.3.1 Chunk seal + STREAM nonce (normative: V2 lock)

Per-chunk content sealing **conforms to SFrame** (draft-ietf-sframe, normative): the per-frame AEAD model with a `(KID, CTR)` header, here bound as `KID = stream_id` and `CTR = counter`, AEAD = AES-256-GCM, keys derived per MLS epoch (the canonical SFrame+MLS composition, CC 5.1). The STREAM nonce below is CEG's binding profile of SFrame's per-sender nonce derivation. Per-chunk content sealing uses AES-256-GCM (NIST FIPS 197 + SP 800-38D). The 12-byte (96-bit) nonce follows the **STREAM** layout (Hoang, Reyhanitabar, Rogaway, Vizár — *Online Authenticated-Encryption and its Nonce-Reuse Misuse-Resistance*, CRYPTO 2015):

```
nonce[12] = prefix[7] || counter_be[4] || last_flag[1]
```

- `prefix[7]` — derived, NOT transmitted: `prefix = HKDF-SHA256(epoch_dek; info)[0..7]` where the HKDF `info` is the **byte-exact** concatenation: `info = b"ciris-stream-nonce/v1"` ‖ `stream_id_utf8` ‖ `epoch_be8`, with `stream_id_utf8` = the UTF-8 bytes of `stream_id` and **`epoch_be8` = the `u64` epoch as 8-byte big-endian** (`epoch.to_be_bytes()`, consistent with `counter_be`). **This encoding is normative and MUST be byte-identical across producer, substrate, and every consumer** — the CC 5.1 zero-trust-of-host posture has consumers recompute this exact nonce to *open* chunks, so any BE/LE/ASCII disagreement on `epoch` yields a different `prefix` → different nonce → GCM auth-tag failure (silent whole-stream decryption failure), the same hazard class `counter_be` already guards. No length-prefix on `stream_id` is required: the fixed-length tag prefix + fixed 8-byte `epoch` suffix make the parse unambiguous (distinct `(stream_id, epoch)` pairs always yield distinct `info` — unequal `stream_id` length changes total `info` length; equal length splits unambiguously). Matches the `KEY_GRANT_V1_INFO` versioned-context HKDF pattern (`b"cewp-key-grant/v1"`). Per-`(stream_id, epoch)` unique; verifiable by any holder of the epoch DEK

- `counter_be[4]` — 32-bit big-endian; hard ceiling 2^{32}-1 chunks per epoch. Substrate MUST force an epoch roll before wrap. Recommended operational cap: `MAX_CHUNKS_PER_EPOCH` = 2^{24} (~16.7M chunks/epoch) to keep per-epoch state + proof sizes bounded (operator-tunable)
- `last_flag[1]` — `0x01` on the final chunk of an epoch (sealed by `seal_stream` per CC 5.1); `0x00` otherwise. The distinct nonce on the final chunk gives **truncation + append resistance**: an adversary cannot drop the final chunk and pass off a short stream, nor append past a sealed segment

Cross-epoch counter reset is nonce-safe (normative reasoning): GCM's catastrophic case is reuse of a `(key, nonce)` pair. On epoch roll the DEK changes, so a reset counter lives in a different keyspace — `(DEK_e, nonce=0)` and `(DEK_{e+1}, nonce=0)` are distinct pairs. The enforced invariant is therefore only within a single epoch: counter strictly monotonic, never wraps (guaranteed by the forced roll). Across epochs, reset is free.

Single-sender-per-`(stream_id, epoch)` invariant (normative — nonce-collision safety; as in SFrame/MLS): the `counter_be[4]` space of a `(stream_id, epoch)` is owned by **exactly one** sender — the `stream_id`'s producer. Two distinct senders MUST NOT seal chunks under the same `(stream_id, epoch)` DEK, or their independently-incremented counters could collide into a reused `(DEK, nonce)` pair (GCM-catastrophic). In **group video** (CC 5.3.3.2) each participant emits their **own** `live_stream` with their **own** `stream_id`, so the nonce prefix `HKDF(epoch_dek; "ciris-stream-nonce/v1" ‖ stream_id ‖ epoch)` is per-sender-unique by construction — the same hazard SFrame avoids with per-sender keys, achieved here by per-sender `stream_id`. Substrate MUST reject a chunk-append whose `(stream_id, seq)` collides with an existing sealed chunk (replay/forking guard, CC 5.1).

5.3.3.2 Realtime group communication: composition

The delivery axis was framed for 1:1 observer-share and 1:N broadcast multicast. Realtime **group communication** — group video, voice, desktop/screen sharing, text chat, and topic-scoped channels with sub-channels — is the **same primitive set at N↔N cardinality**. It composes entirely from `community` (CC 3.2) + `live_stream` (the CC 5.3.3 streaming surface) + `chat_message` + member/transport resolution (CC 4.4.3.2.4.1). **Zero new structural primitives** — this is the thirteenth path on the CC 1.7 claim and the most product-complete: the whole realtime-collaboration surface is composition.

The composition map (all generic — no new wire):

Surface	Composes from
1:1 / group video call	each participant emits a `live_stream` (their A/V) scoped to the channel `community`; each subscribes to peers. N↔N = N simultaneous bidirectional streams over a small roster
Voice channel	identical to group video with an audio-only codec; same `live_stream` wire
Desktop / screen sharing	a `live_stream` whose source is a screen capture (1→N within the channel); same chunk-seal + epoch-DEK wire
Text chat	`chat_message` at `cohort_scope: community`, `community_id: <channel>`; threads via `topical_relation: replies_to`
Channel	a `community` (persistent) — its roster gates who can join the call / read the chat

Surface	Composes from
Sub-channels	nested `community` membership (CC 4.4.3.2.5 multi-level pattern): a parent "space" community whose members admit child channel-communities; sub-channel members are a subset of the space roster
Presence ("who's here")	the D6 reachable set (CC 5.3.3.4) — node-local, never an attestation, never logged
Invite / join / leave	community admission ceremony (CC 4.4.3.2.3) — invite = membership proposal (`membership:proposal:v1`); join = admitted member, by a membership widening **on the invitee's own** `membership:acceptance:v1` (CC 3. 1.3.2); decline = `membership:decline:v1`; leave = forward-only `withdraws`

Transport profiles: *(normative; extends CC 5.3.3.5)*

Profile	Topology	Transport
Broadcast (1:N, large N)	asymmetric	pull-only `ContentFetch` (CC 5.3.3.5 E2); relay tree → 1.x
Realtime small/medium group (calls, huddles, ≤ ~50)	symmetric mesh	**direct Reticulum Links between participants** (low-latency push; no relay tree needed at small N)
Realtime large group (≫50 in one A/V room)	fan-out	selective-forwarding relay (SFU role) → 1.x (same relay-tree axis as broadcast scale)

The low-latency realtime profile (direct Reticulum Links) is **in 1.0 scope** because small/medium rosters need no relay tree — RNS Links give encrypted low-latency point-to-point natively. The wire is identical to broadcast (chunk-seal CC 5.3.3.1, per-`(stream_id, epoch)` epoch DEK CC 5.1, per-stream STH CC 5.3.3.3); only the transport profile + roster size differ: a live chunk and a stored chunk are the **same sealed object**, so "record a call" is retention policy over chunks that already exist as a DAG, "VOD" is the puller at offset zero, and a licence is a `key_grant` the owner chooses to issue over the same bytes. **PQC is unchanged and mandatory** — epoch DEK wrapped `wrap_algorithm: v2` (X25519+ML-KEM-768) at rest, hybrid KEX in transit.

Ephemeral vs persistent rosters: a persistent channel is a long-lived `community`; an ad-hoc call is an **ephemeral** `community` (short `valid_until`, torn down on last-leave). Same primitive, different lifetime — no special-case "session" primitive.

The breadth confirmation: a full realtime group-communication platform — group video, voice, screen sharing, chat, and channels-with-sub-channels — requires **no addition to the 1+4 structural set** beyond the CC 5.3.3 delivery axis and the `community` membership machinery already locked.

5.3.3.2.1 `codec_id` namespace: realtime A/V chunk codec discriminator (normative)

CC 5.3.3.2.2 makes the realtime chunk payload codec **application-defined and opaque to the substrate**. For realtime **A/V at scale**, a hop fanning out chunks must drop chunks for per-receiver bandwidth degradation **without** decrypting them — so the codec's layer-numbering semantics must be a **clear (non-AEAD) discriminator**. CEG defines a 1-byte `codec_id` namespace so every implementation reads the same meaning. **This is a namespace on the transport-layer chunk header, not a change to the CC 2.1 attestation envelope** — the frozen 1+4 surface and its canonicalization are untouched.

Wire position (normative). `codec_id` (1 byte) + `ChunkLayer { spatial: u8, temporal: u8, quality: u8 }` (3 bytes) = a **4-byte additive block** at `SealedAvChunk` header offset **48..52**, after the 48-byte fixed header. **Clear metadata, NOT inside the AEAD** — a relay drops chunks by `codec_id`/`ChunkLayer` without touching the inner epoch-DEK seal (CC 5.3.3.1); tampering causes mis-decode or drop, never a crypto break. No length-prefix is needed — the block is fixed 4 bytes at a fixed offset.

Hex	Codec	Semantics
`0x01`	**AV1 SVC**	Scalable Video Coding — 3 spatial × 4 temporal × N SNR layers; base layer required to decode anything. Production default (WebRTC-native, royalty-free). The deployable codec today.
`0x02`	JPEG XS (layered)	Low-latency intra-only; broadcast use case. **Reserved.**
`0x03`	**Symmetric MDC**	Multiple Description Coding — any subset of chunks decodes at proportional fidelity, no base-layer floor. **The substrate design target** (below). **Reserved** — encoder lineage academic-grade today; substrate is MDC-ready.
`0xFF`	Opaque	No scalable-coding semantics. `ChunkLayer` MUST be `{ 0, 0, 0 }`. Default for legacy / non-layered streams.
`0x04`–`0x7F`	—	**Reserved** for future standardized codecs (CEG-assigned).
`0x80`–`0xFE`	—	**Experimental / per-deployment** — no cross-federation meaning guaranteed.

`codec_id` lets a receiver know whether `ChunkLayer { spatial: 2, temporal: 2, quality: 0 }` means "SVC base + 2 spatial enhancements" or "MDC quadrant" — without it, layer numbers are ambiguous across codecs. The substrate (Edge) never picks the codec — choice is upstream (agent/server tier); Edge needs only the namespace stable. The variable-depth `SubStreamPath` (MDC tree-path encoding) is a follow-on.

MDC-primacy design intent (informative). The user-facing contract CEWP realtime A/V targets is *"any node can request a lower-bandwidth stream from peers — as simple as taking every other chunk, down to a blinking dot."* MDC (`0x03`) matches that **symmetrically**: drop any subset → decode the rest at proportionally lower quality, no coordination, no base-layer floor. SVC (`0x01`) is production-deployable today but has a floor (base-layer bytes must arrive) and needs coordination for an "every other chunk" drop. The substrate is **MDC-shaped** (the `ChunkLayer` / `SubStreamPath` model is symmetric-drop-ready) even while production streams ship SVC; the ~20–40% MDC compression overhead vs SVC at equal quality is the accepted cost of symmetric drop semantics.

5.3.3.2.2 Realtime non-A/V data streams (normative scope boundary)

The realtime profile is **not media-only.** Any high-frequency mutable shared state — multiplayer game ticks, collaborative-editing operations (CRDT/OT), live cursors/whiteboard strokes, remote-control input, high-rate telemetry — rides the **same** CC 5.3.3.2 realtime transport (direct Reticulum Links / SFU at scale) with an application-defined payload codec in place of an A/V codec. The wire is identical: per-`(stream_id, epoch)` epoch DEK (CC 5.1, `wrap_algorithm: v2` PQC), STREAM-nonce chunk seal (CC 5.3.3.1), per-stream STH (CC 5.3.3.3). A data-stream chunk is just a chunk.

Scope boundary (the explicit call):

- **IN scope (transport):** ordered, sealed, authenticated, PQC-encrypted realtime delivery of arbitrary data-stream payloads — covered by CC 5.3.3.2, no addition.
- **OUT of scope (merge semantic):** the conflict-free / convergent-merge logic for shared mutable state (CRDT, Operational Transform, last-writer-wins, etc.) is **application-layer**, not a CEG primitive — consistent with the CC 1.13.4 discipline ("the substrate stores; the wire transports; CEG describes the shape of the claim; consumer policy composes verdicts"). CEG carries the ops; the application converges them. A future codified merge primitive would route through the CC 4.5.1 amendment process if real demand pulls it (the downstream-demand-pulls-additions discipline), but it is explicitly NOT required for 1.0 — realtime collaborative apps are buildable on the transport today.

5.3.3.2.3 `SealedAvChunk` wire layout (normative)

The realtime A/V chunk that lands on each RNS Link payload (and the broadcast pull path). **Byte layout (normative):**

```
offset  field                      encoding
0..32   stream_id                  32 bytes (caller-derived: sha256(stream_meta))
32..40  epoch                      u64 big-endian
40..48  chunk_seq                  u64 big-endian
48..49  codec_id                   u8  (S10.5.8.2 namespace)
49..50  layer.spatial              u8
50..51  layer.temporal             u8
51..52  layer.quality              u8
52..    double_sealed_ciphertext   remaining bytes
```

- `CHUNK_HEADER_LEN = 48` (the `stream_id`+`epoch`+`chunk_seq` fixed header); `CHUNK_CODEC_LAYER_LEN = 4` (the `codec_id`+`ChunkLayer` block).
- **The block is always present; no length-disambiguated compatibility (normative).** The 4-byte `codec_id` + `ChunkLayer` block is part of every chunk: the header is 52 bytes, and a wire shorter than 52 bytes, or whose ciphertext does not carry its AEAD tag, is refused as malformed. Length does not select a layout: a 48-byte wire carries no ciphertext and no tag and could never open.
- `codec_id` + `layer` are **clear metadata, NOT inputs to the AEAD** (CC 5.3.3.2.1) — a relay drops by `(codec_id, layer)` without compromising the inner DEK; tampering causes mis-decode or drop, never a crypto break.

5.3.3.2.4 `ChunkLayer` + `ReceiverLayerPolicy`: SVC layer model (normative)

`ChunkLayer` is the 3-byte SVC layer descriptor in the chunk header (`spatial`, `temporal`, `quality`, each `u8`). Each axis is **monotonic**: layer `0` is the base (lowest fidelity, always required); each increment is an additive enhancement. A receiver reconstructs from the prefix `0..=max_spatial` × `0..=max_temporal` × `0..=max_quality` of cells. The base cell `{0,0,0}` is the **"blinking dot"** — the minimum a participant can subscribe to. For `codec_id = 0xFF` (opaque) the layer MUST be `{0,0,0}`.

`ReceiverLayerPolicy { max_spatial, max_temporal, max_quality }` (each `u8`) is the per-receiver drop policy. It is **advertised over the existing `federation_session` / `key_grant` entitlement surface — NOT a new wire** — and the sender drops chunks above the cap without re-encoding. `admits(layer)` is the per-axis test `spatial` $\leq$ `max_spatial` $\wedge$ `temporal`

$\leq$ `max_temporal` $\wedge$ `quality` $\leq$ `max_quality`; a chunk tagged `codec_id = 0xFF` MUST be admitted **unconditionally** regardless of policy (the fan-out filter short-circuits before consulting `admits`). Canonical policies: `BLINKING_DOT = {0,0,0}`, `UNCAPPED = {255,255,255}`. This composes with the inner-once / outer-N fan-out optimization: the inner seal runs once per chunk; the outer seal runs only for the `(receiver, chunk)` pairs the policy admits.

5.3.3.2.5 Double-seal + deterministic nonce derivation (normative)

`double_sealed_ciphertext` is **outer-AEAD(inner-AEAD(chunk_plaintext))** — two independent AES-256-GCM layers (12-byte nonce + 16-byte tag, standard ring layout each). The **inner** seal is end-to-end (the epoch-DEK content seal); the **outer** seal is the per-RNS-Link transit wrap (a relay sees the outer layer only, never plaintext — the CC 5.3.3.5 E1 two-layer posture). Both nonces are **deterministic** (no nonce is transmitted — every holder recomputes; collision-safety rides the CC 5.3.3.2 single-sender-per-`(stream_id, epoch)` invariant).

One chunk seal (normative). The inner seal **is** the CC 5.3.3.1 chunk seal: the STREAM nonce `prefix[7]` $\|$ `counter_be[4]` $\|$ `last_flag[1]`, with `counter` = the chunk's `chunk_seq` (the header's `u64` field carries it; it MUST stay below 2^{32} within an epoch, the 5.3.3.1 ceiling, with the forced epoch roll before wrap) and `last_flag` set on the epoch's final chunk. There is no second inner nonce: the label `CIRIS-AV-INNER-V1` is **retired**, so a live chunk is stored as it was, a stored chunk can be served into a call, and CC 5.3.3.4's reconnect-then-pull works for a call. Live and stored chunks are one object model, differing only by subscriber offset (as in IETF Media over QUIC, CMAF, and MLS + SFrame, which 5.3.3.1 already cites). The **outer** nonce is:

```
outer_nonce = SHA-256( b"CIRIS-AV-OUTER-V1" || link_id || link_seq_be8 )[0..12]
```

The label bytes (`b"CIRIS-AV-OUTER-V1"`, ASCII, no terminator) are a **domain separator pinned by this section**. `link_seq` is `u64` **big-endian** and monotonic per RNS Link (transit replay guard). **Conformance is proven by the vector set** for the outer nonce and `to_bytes`, published with CIRISVerify; the inner seal's vectors are 5.3.3.1's. `codec_id` and `ChunkLayer` stay clear metadata outside both AEADs.

5.3.3.3 Per-stream log + stream-root (normative: V1 lock)

A live stream is its own transparency log, which is what makes a producer accountable for what it streamed: the signed roots commit the producer to a single chunk history. For each live_stream:

- `log_id = stream_id`; chunks = leaves; stream-root = `SignedTreeHead{ log_id: stream_id, tree_size: chunk_count, root_hash, timestamp, signature }`. **A live stream produces these at the cadence below whether or not anyone records it**.
- **Producer signs the STH — MANDATORY** authenticity root; hybrid Ed25519 + ML-DSA-65 per the CC 5.3.1 `signing_bytes` discipline; canonical bytes per CC 2.6.1 JCS for the envelope-bearing wrapper
- **Witness cosign — OPTIONAL**, via the CC 5.3.1 path verbatim. This is the best-effort / accountable split:
- **Best-effort** (open media) → producer-signed root only

- **Accountable** (paid media, registry propagation, emergency) → witness cosign per CC 5.3.1.1 consistency-proof, which is the **anti-equivocation guarantee** — producer cannot show different chunk-K to different subscribers nor rewrite mid-stream
- **Cadence**: producer publishes a signed root every `K` chunks OR `T` seconds (whichever first), **always at an epoch boundary** (CC 5.1) + at `sealed_at`. Witness cosign runs **per-epoch** (coarser than per-K to keep cosign-quorum cost off the hot path). Default pins: **K=64, T=2s** (operator-tunable)
- **Incremental verify** (D4): each leaf's "root-after-K" lets a subscriber verify chunk K's inclusion against the nearest signed STH $\geq$ K via the CC 5.3.1.1 consistency path. No new commitment structure beyond the per-leaf chain-link + periodic STH that CC 5.3.1 already provides.
- **Accountable-stream quorum**: Policy E (CC 4.4.3.1 locality-scaled quorum) applies — not a fixed N. Emergency-channel roots want a higher quorum than a paid-media stream; locality scaling provides the gradient.

5.3.3.4 D6 liveness invariant: entitled vs reachable (normative)

Two sets are NEVER conflated — the durable record of who is *allowed* in, and the ephemeral fact of who is *currently here*. Mixing them would turn presence into a logged, federated attestation, which the fail-secure posture forbids:

- **Entitlement roster** (Persist-owned): signed CEG envelope, Edge-propagated, durable, logged. It's **evidence** — it MUST propagate + be auditable. Per CC 4.4.3.2 Policy M community-membership composition
- **Live-reachability set** (Edge-owned): generalizes the holding claim's 72 h liveness (CC 3.1.3.3) down to seconds-to-minutes for live-multicast. Node-local presence tracker. **NEVER an attestation, never a holding claim, never replicated, never logged**

Fan-out invariant: `fan_out(C) = entitled(C) ∩ reachable(now)`.

Heartbeat-suppression discipline: this is a **producer-side-refusal invariant** (same class as the CC 5.2 no-holding-claim-leaves-a-cohort rule). Missed (entitled-but-unreachable) members fall back to pull on reconnect — substrate does NOT keep retrying push, does NOT emit a "delivery_failed" attestation, does NOT log liveness state. The reconnect-then-pull catch-up rides CC 5.1 `history_on_join`; under the one chunk seal (CC 5.3.3.2.5) the pulled chunks open with the same nonce as the pushed ones, so it works for a call.

5.3.3.5 Transport: Edge layer (normative — E1–E4 lock)

Decision	Behavior
E2 — pull-only	Multicast is **pull-only**: producer seals chunks under the epoch DEK → emits holding claims at the stream's scope → subscribers pull via the existing `ContentFetch` path. Relay / fan-out tree → 1.x

Decision	Behavior
E1 — two-layer crypto (security-critical)	Transit-key is a **hop-by-hop transport wrap UNDER the E2E epoch DEK** (two independent crypto layers). MUST NOT replace the cascade — a relay never sees plaintext. Transit-key is for path-confidentiality; epoch-DEK is for end-to-end content confidentiality. **PQC in transit — the layered posture (normative).** The RNS link layer is classical (ephemeral X25519, Ed25519, AES-256-CBC + HMAC) and is **not relied on for post-quantum confidentiality**: the RNS link request is a fixed X25519‖Ed25519 layout, and a link-level ML-KEM would break interoperability with Python RNS. The guarantee is carried above it, in three layers, all of them required: every content key is wrapped X25519 + ML-KEM-768 (the at-rest cascade, `wrap_algorithm: v2`, CC 5.1); every federation record is signed Ed25519 + ML-DSA-65 (CC 2.6.1.2); and **every byte an edge sends over an RNS link MUST be under a hybrid transit seal keyed by an X25519 + ML-KEM-768 exchange bound to that link's `link_id` and to a transit pair** authenticated to the identity the initiator means to reach **(that identity's own hybrid-signed binding, a Rooted binding, or a pairing pin — never a bare or unsigned announce).** The pre-seal route fetch: **a dialler that holds no pair for the node it means to reach — two strangers on a bare mesh, each holding only the other's announce, which cannot carry a post-quantum pair — opens the link and sends, as its first frame, a** route request **naming the dialled key and saying whether it holds that key's record; the responder answers with its own hybrid-signed `transport_destination` row (and its own key record, when the dialler says it holds none) and nothing else, and only for its own key, from its own live row carrying a pair. A stranger cannot check a row's hybrid signature without the signer's public record, which an announce does not carry; the key record is the same self-signed public row the key plane replicates to everyone, so sending it discloses nothing more. The dialler checks that the signer and subject of each row equal the dialled key before it stores anything, admits the key record through the key door (proof of possession and the CC 2.6.8 fingerprint), admits the route row through authenticated admission, verifies it under the rule above (fingerprint, in-row separation, signature), and only then runs the handshake; a row signed by or about another key is refused and never stored. The responder answers a route request at most once per link and rate-limits route requests node-wide before parsing them past their tag. That public, self-authenticating row is the one thing besides the handshake that crosses a link unsealed: it is the row a relay would hand over and an announce already makes public, so the round trip discloses nothing an announce does not; the receiver order (per-sender, per-link and node-wide rate) runs before it, the responder serves nothing and buffers nothing until the seal completes, and a link whose first frame is anything else is dropped. Where even that fails — no reachable responder — the pairing pin (CC 3.2 adding a device, (f)) is the failover, the only path for two nodes that cannot exchange a single public frame. With that, there is no classical-only fallback for any content and no advisory phase: a node that does not run the seal cannot talk to the mesh until it upgrades; and there is no public classical lane — a standard LXMF client (Sideband and the like) that cannot run the seal is outside the mesh and is not served by an edge's LXMF path** — the CC 3.3.6.2 transit KEX, run once per link as its first frame and covering every frame that follows (replication, membership, first contact and the device-join request, chat row headers, the A/V outer seal alike), not only the A/V hop. On the first-contact plane the initiator has no owner yet; the seal still binds its self-signed node key, never an anonymous one (CC 3.2, adding a device, (e)). The transit-key wrap of this row is that seal. Classical-only transport is rejected for streaming. The in-transit **KEX** identifier names a *different* construction from the at-rest `wrap_algorithm` and carries its own identifier string; the two MUST NOT be compared, substituted, or accepted in each other's field (CC 5.1 one-construction-one-identifier rule). **Which key binds a non-member relay:** the relay's transit KEX pair published on its signed `transport_destination` binding (CC 3.3.6.2), static against the initiator's ephemeral; never a KeyPackage init key, never an ephemeral-only exchange; a node with no valid transit pair cannot relay.

Decision	Behavior
E3 — fan-out = entitled ∧ reachable	**Persist owns durable entitlement** (the roster: signed CEG envelopes, replicated, logged). **Edge owns transport-reachability** via a node-local presence tracker. Fan-out targets the intersection. Reachability is NEVER an attestation, never a holding claim, never replicated, never logged — consistent with the CC 5.2 'cohort_scope: self\
E4 — durable side rides existing federation-attestation path	Durable entitlement (roster + epoch-key grants) rides the **existing federation-attestation Edge path** — just more `federation_attestations` rows. NO net-new Edge transport for the durable side. Net-new is only on CC 5.3.3.3 streaming-log endpoints

5.3.3.6 Delivery receipts (normative: D5 / V3 lock)

A `delivery_receipt:{stream_id}` Contribution is a subscriber's signed acknowledgement that they received chunk K under the named stream + epoch. **Best-effort default**; opt-in for **accountable** profiles (registry propagation, emergency).

Canonical bytes (domain-separated + length-prefixed, matching `SignedTreeHead::signing_bytes` discipline; per CC 2.6.1 the envelope-bearing wrapper is JCS-encoded, but the receipt's signed-bytes inner payload follows the explicit-length-prefix shape below):

```
receipt_signing_bytes =
    "ciris-delivery-receipt/v1"                       // domain separator
  || len(subscriber_key) || subscriber_key
  || len(stream_id)      || stream_id
  || epoch        (u64 LE)
  || chunk_root   ([u8; 32])
  || K            (u64 LE)
```

Both `epoch` (key-rotation index — for per-epoch entitlement / billing) and `K`/`chunk_root` (chunk position + its committed root) are independent indices — both required (each names a distinct authorization scope).

Verify check is a JOIN, NOT a sig-check:

1. **Signature valid** over the canonical bytes (subscriber hybrid Ed25519 + ML-DSA-65 sig). Necessary but **not sufficient**.
2. **`chunk_root` is a real published STH root** — MUST equal a `SignedTreeHead.root_hash` actually published for `log_id = stream_id` at `tree_size` $\geq$ `K`. A phantom / self-invented root → REJECT. For accountable streams, "published" means **witness-cosigned** (the CC 5.3.1 path), so the subscriber cannot collude with the producer on a private root.
3. *(Recommended for accountable)* **Inclusion proof** chunk K → `chunk_root`. Upgrades the receipt from "subscriber saw a root" to "subscriber saw a root that provably commits to chunk K".

Semantics — proof-of-DELIVERY, not proof-of-CONSUMPTION: the receipt proves the subscriber received bytes committing to chunk K. It does NOT prove they decrypted those bytes (they may not hold the epoch DEK). Consumers MUST NOT overclaim a delivery receipt as proof of consumption. Per the `MISSION.md` fail-honest invariant + CC 1.7 "Verify authenticates origin, does not compose 'delivered'/'owes N'", Verify's role is validation-not-adjudication: emit the validated receipt as an attestation on `delivery_receipt:{stream_id}` — the "delivered" verdict is consumer policy.

Accountable-stream receipt quorum: Policy E (CC 4.4.3.1 locality-scaled) — same shape as the CC 5.3.3.3 STH quorum.

5.3.3.7 Framing (normative)

A stream is **its own per-stream transparency-log instance** (`log_id = stream_id`). A `live_stream` MUST NOT append chunks into the federation provenance log (CC 5.3.1's global log carrying builds / licenses / identities) — millions of media chunks would pollute provenance and inflate the global tree. The CC 5.3.3 path **reuses** the CC 5.3.1 `SignedTreeHead` / `ConsistencyProof` / `WitnessConsistencyProof` / cosign abstractions, instantiated per-stream as separate log instances under the same RFC 6962 algorithm.

The 1+4 wire-format lockdown holds: there are no new `attestation_type` values. Stream chunks ride content addressing; stream-roots ride the existing `SignedTreeHead` shape; delivery receipts ride `scores` against the new `delivery_receipt:{stream_id}` reserved prefix (CC 3.4.6).

5.3.3.8 What the streaming surface documents

Scope pointer: the CC 5.3.3 streaming surface — delivery axis (CC 5.3.3.3–CC 5.3.3.4) — does **not** change the 1+4 primitive set (CC 2.4) (delivery rides existing primitives), does **not** bundle the streaming-half substrate impl (the streaming-chunk store + the parallel CHECK-arm migration), keeps push-mode multicast relay/fan-out pull-only in 1.0, and leaves the K / T / MAX_CHUNKS_PER_EPOCH constants + accountable-stream quorum operator-tunable.

5.3.4 Trust-root discovery: the GenesisBundle + accord holders

These endpoints publish the trust root itself — the accord-authorized GenesisBundle, and the accord holders it names — so a verifier can discover *whom* to trust before trusting anything. A consumer MUST verify the bundle's quorum against its own records before promoting any field to a trust root: this is the fidelity / fail-secure floor for the whole discovery surface. Nothing a serving install signs on its own account is a root.

`GET /v1/trust-root/bundle` — the portable trust root (`/v1/steward-key` is a kept alias)

Returns the **GenesisBundle**: the charter, the accord-holder roster, the scopes and the serve-node grants, hybrid-authorized by the accord holders at the genesis ceremony. A registry publishing its own key as its own root, `hardware_class` self-attested, is the weakest form of a producer claim (CC 4.2.2.1); the registry is a conferred slice of a canonical node, not a root. The same body is served at both paths.

Response (`200 OK`):

```
{
  "bundle": {
    "version": 1,
    "family_key_id": "humanity-accord",
    "holders": [ "<SignedKeyRecord>", "..." ],
    "serve_nodes": [ "<SignedKeyRecord ciris-canonical-1-...>" ],
    "consensus_protocol": "quorum:2/3",
    "attestations": [ "<SignedAttestation: charter(s), lifecycle, grants, the ciris-canonical birth record
     >" ],
    "authorizations": [ "<GenesisAuthorization A1>", "<GenesisAuthorization B1>", "..." ],
```

```
    "produced_at": "<rfc3339_canonical>"
  },
  "bundle_fingerprint": "sha256:<64-char-lowercase over the JCS-canonical bundle>",
  "charter_root_key_id": "<read off the bundle; verify it>",
  "served_by": { "node_key_id": "<relaying node>", "accepts_this_root": true }
}
```

Authority lives inside `bundle` and nowhere else (normative). The bundle is **tamper-evident, not authentic** (CC 3.2 T5): its holders' hybrid authorizations over the charter prove the bundle is internally consistent, and an attacker's self-consistent bundle under the same identifiers verifies identically. A consumer therefore MUST hold an **out-of-band anchor before accepting a bootstrap bundle** — either the pinned `bundle_fingerprint` (the `sha256:` over the JCS-canonical bundle, published with the release and baked into conformant builds) or the pinned fingerprints of the accord-holder keys — and MUST compare the received bundle against it at attach time; `ciris-canonical` and `humanity-accord` are **names, not anchors**, and a consumer with no anchor MUST NOT promote anything in the response to a trust root. With the anchor held, the consumer re-derives the quorum from its *own* records (`verify_bundle_quorum` — the authorizations counted against the roster the consumer holds or has just pinned), and from then on the root is valid until revoked (T4). Everything outside `bundle` — the served fingerprint, the charter root id, `served_by` — is unsigned convenience metadata the consumer recomputes (the fingerprint against its pin) or ignores; the served `bundle_fingerprint` is never itself the anchor. There is deliberately **no `response_signature`**: signing the wrapper would prove only that the relaying node said it, which is worthless as a root, and it would invite consumers to check the envelope instead of the artifact. `hardware_class`, `signature_mode` and `threshold_policy` MUST NOT appear on the outer envelope; a holder's custody class rides its signed key record and is corroborated, or not, per CC 4.2.2.1. `served_by.accepts_this_root: false` is a legitimate state — a node may relay a root it has not accepted (CC 3.2 T3) — and is the operator's un-trust lever made visible.

A consumer pins the community anchor `community_key_id: ciris-canonical` and the accord family `humanity-accord`, resolves the live member set via `resolve_community` (CC 4.4.3.2.4), and never hard-pins a serving install's fingerprint. A root is valid until revoked, never until a timer lapses (CC 3.2 T4); what freshness gates is *attaching*, under T4a, and the object that carries it is served beside the bundle.

The lineage head beside the bundle. The genesis head of each lineage — the `humanity-accord` family record and the `ciris-canonical` community birth record (CC 3.2 T6: the signed roster record at a version is the head) — and the charter that carries each root's `attach_window_secs`, `witness_cadence_secs`, `witness_quorum` and `witnesses[]`, are members of `bundle.attestations` and are therefore pinned by `bundle_fingerprint` with everything else in it. The route serves, outside `bundle`, a `lineage_head` member: the **head chain since genesis** of each lineage — every version of the signed roster record, the charter bytes each names, and the cosignatures collected over each under CC 5.3.1's lineage-head domain — and the same chain is served at `GET /v1/trust-root/lineage-head` (with `?since=<digest>` for a consumer refreshing before it attaches), any object in it fetchable by digest. It is unsigned convenience like everything outside `bundle`: the consumer walks `prev_head_digest` from the genesis head it holds from the bundle, checks each head's quorum against its **parent's** fold and each cosignature against its **parent's** directory (never the head's own `charter_digest`, which a fork head chooses), counts only independent witnesses, and applies CC 3.2 T4a — fewer than `witness_quorum` is `trust_root_head_unwitnessed`, a newest counted cosignature older than `attach_window_secs` is `trust_root_head_stale`, and in either case the consumer fetches a fresher head or refuses; it never attaches on the stale one. A consumer SHOULD obtain at least one counted cosigna-

ture from that witness directly. **Shipped charter defaults.** The shipped roots are minted with **witnessed mode off** (`witness_quorum: 0`, no directory; CC 3.2 T6) and `attach_window_secs` **604800**; attachment is by the pinned `bundle_fingerprint`. The accord's charter carries `halt_fuse_secs` **86400** (24 hours, CC 4.2.1.1). When witnessed mode is turned on by an amendment signed by a strict majority of the standing accord roster, the defaults are, for both `ciris-canonical` and `humanity-accord`: `attach_window_secs` **604800** (7 days), `witness_cadence_secs` **86400** (24 hours), a directory of **exactly three** independent witnesses and therefore `witness_quorum` **2** ($\lfloor 3/2 \rfloor + 1$ — a fourth witness raises it to 3, since two of four is not a strict majority); all four are charter members, changed only by re-scrub, and the window exceeds the cadence plus skew as T4a requires. Once attached, T4 governs: the served head's age is a liveness signal on the trust surface, never an expiry.

The bundle is the Rooted seed for transit pairs (normative; `GENESIS_BUNDLE_VERSION` 4). Under the per-link hybrid seal of CC 5.3.3.5 E1 a node dials only a peer whose transit pair it holds **authenticated to the identity it means to reach** — that identity's own hybrid-signed `transport_destination` row (CC 3.3.6.2), from any source; or a Rooted binding; or a pairing pin (CC 3.2 (f)) — and such a row otherwise arrives over a sealed link, so a device that knows no one needs a first row from somewhere, and for the seated serve nodes it is this bundle. **A canonical node is one source of route rows, never a required party** (the premise of this document is a network with no company in the middle, and a trust decides what a node serves, CC 3.2 T8, not whether two keys may speak privately): two peers holding each other's signed route rows — from a contact code, a mutual peer, or any relay of the bootstrap kinds — seal, exchange the allegiance facts and root each other directly. The bundle carries, in the member **`serve_transport_destinations`**, the seated serve nodes' signed `transport_destination` rows (CC 3.3.6.2), each carrying its transit pair; the member is inside the `authorization_digest` preimage, so the holders' quorum authorizes the exact set as it authorizes `serve_nodes`. At bundle verification each row MUST be signed by a serve node listed in the bundle's `serve_nodes` and MUST carry a pair that passes the in-row separation; any failure fails the bundle closed. At first open the rows are seeded through the ordinary transport-destination admission and land Rooted, so a fresh node with no network holds a canonical node's pair, dials it first under the seal, and replication brings it the mesh's announced bindings. A version-4 bundle is a trust-root re-mint, and it comes in two forms. A **fresh genesis** is the final-genesis ceremony: every holder signs the charter and then the genesis records and the authorization, because only each holder can give their own recovery commitment and successor seat and the founders sign the `ciris-canonical` birth — unanimity is what that ceremony gathers, and any change to the charter or the roster takes it. An **amend** carries the pinned, verified bundle's charter, grants, lifecycle row, serve-node records, both genesis records and `produced_at` forward **byte for byte** — every carried signature stays valid over its own unchanged envelope — adds `serve_transport_destinations`, and signs only the new `authorization_digest`; it finishes at **M distinct holder authorizations** (the charter's `quorum:2/3`, CC 4.2.3; verification requires `max(M, strict majority of the roster)`, which is 2 of 3) and refuses if any carried byte differs from the pinned input, so an amend cannot smuggle a changed charter or grant. Seeding the canonicals' transit pairs is the same class of act as blessing a canonical, already a strict-majority act, so two holders gain no power they lack. The v3 and v4 bundles share `produced_at` and differ by version and digest. There is no separate propose/cosign path; the amend is a mode of the final-genesis assembler. An earlier pin keeps working but seeds no pairs. The rows the bundle carries are **signed** `transport_destination` rows, each the node's own signed self-emission. **The first seed is emitted offline:** under the seal no existing node can dial a canonical whose pair it does not hold, and the re-mint cannot read a row the canonical has not emitted, so the cycle is broken on the host — a seated canonical's operator emits the node's hybrid-signed row, pair included, from the node's own custody with

no network (the transit pair is derived deterministically from the custody seed, so the offline row and the row the node emits live after upgrade carry the same pair); the ceremony re-mints from those rows; persist, edge, server and client releases then ship the re-minted bundle with the sealed transport and upgrade together. Bundle verification checks each row's signature and its signer's seat, never its age (CC 3.2 T4: valid until revoked), so a row emitted before the ceremony seeds a node opened long after; the node's live row supersedes it on the first sealed link, same pair, later `asserted_at`. **Overlap rule:** a seated serve node MUST keep the pair the pinned bundle carries valid until a re-minted bundle carrying its new pair has propagated, or make the rotation itself a re-mint; the node's own rotation door enforces it — a rotation that would retire the pinned pair before its replacement is Rooted on every node is refused. **A rotation is a quorum act, not a re-mint:** the holders' quorum re-binds a canonical's destination and transit pair by the accord-signed address update of CC 3.3.6.2, which lands as a Rooted `transport_destination` superseding the bundle-seeded pair on every node that holds the root; the re-mint is needed only to change what a *fresh* node seeds, and a fresh node that seeds the old pair follows the supersede chain to the new one once it has sealed to any canonical. The bundle is the only seed before the first sealed link; the one other way a transit pair reaches a node before any sealed link is the owner's pairing payload at first contact (CC 3.2, adding a device, (f)), pinned for that link alone.

`GET /v1/accord-holders`

Three named holders with hybrid pubkeys + per-holder `hardware_class` + `provisioned` flag. A holder not yet provisioned carries a placeholder fingerprint and `provisioned: false`; consumers MUST NOT honor `constitutional` invocations against placeholders. The holders' authority is the bundle above — this route is a projection of `bundle.holders`, and a consumer cross-checks it against the bundle rather than trusting the projection.

`GET /v1/accord/holders`

UI wrapper around `/v1/accord-holders` with per-holder `accord_emissions[]` for UI rendering. The same projection rule: **no response signature** — the wrapper carries no authority, a consumer cross-checks every holder it shows against `bundle.holders` under the anchor it already holds, and `accord_emissions[]` are the holders' own signed rows, verified individually.

`GET /v1/rotation-history`

Chronological rotation events from `registry_signing_keys` table.

5.3.5 Other Registry endpoints

`GET /v1/builds/{version}` returns the BuildRecordResponse with a `federation_provenance` block. `GET /v1/verify/build-manifest/{project}/{version}/{target}` (Path B) returns the verbatim signed BuildManifest. `GET /v1/agent_files/{kind}?platform_or_target=...` returns the CC 4.4.3.7 trust-composition layers. `GET /v1/partner/{key_id}` composes Profile-Scorecard data from existing tables.

Full response schemas for these endpoints land in the Rust handlers + OpenAPI export; the spec commits to publishing a versioned OpenAPI spec alongside this document.

5.3.6 Common response shape

All CEG endpoints return:

- **Content-Type**: `application/json` (`Accept: application/json` honored; other types respond `406 Not Acceptable`)
- **CEG-API-Version header**: `CEG-Version: <current spec major.minor>` on every response (track the README `Version:` field); clients SHOULD echo `CEG-Accept-Version: <pinned-version>` on request, naming the version they were built against. Per CC 2.6.4 SemVer policy, MAJOR mismatch is a wire-incompat reject; MINOR mismatch is compatible (clients MAY warn).
- **Time-Source header**: `X-CEG-Server-Time: <rfc3339_canonical>` per CC 2.6.2 for client clock-skew bounds
- **Pagination** (where applicable): `?cursor=` + `?limit=` query params; response includes `next_cursor` (null if exhausted) and `total_estimate` (server's best estimate, may be approximate)

5.3.6.1 Error envelope

All error responses MUST conform to:

```
{
  "error": {
    "code": "<ENUM_VALUE>",
    "http_status": <int>,
    "message": "<human-readable>",
    "request_id": "<server-assigned>",
    "details": {<error-specific fields>}
  }
}
```

HTTP status	Error code	Meaning
400	`MALFORMED_REQUEST`	Invalid JSON, missing required field, bad field type
400	`CANONICAL_BYTES_VIOLATION`	Date-time / hex / encoding doesn't match CC 2.6.2 / CC 2.6.3
401	`UNAUTHENTICATED`	Bearer token missing or invalid (admin endpoints)
403	`RESERVED_PREFIX_VIOLATION`	Producer attempted to emit under a reserved prefix without authority per CC 3.4
404	`UNKNOWN_WITNESS`	Witness key_id not registered in directory (CC 5.3.1)
404	`NOT_FOUND`	Generic resource not found (build, partner, key)
409	`IDEMPOTENT_CONFLICT`	Replay detected (e.g., duplicate `(tree_size, witness_key_id)` cosignature with different signatures)
422	`SIGNATURE_VERIFICATION_FAILED`	Ed25519 or ML-DSA-65 failed to verify; `details.algorithm` names which
422	`CLOCK_SKEW_VIOLATION`	`signed_at` exceeds CC 2.6.7 ±5 minute tolerance
422	`WITNESS_QUORUM_NOT_MET`	Insufficient cosignatures to validate

HTTP status	Error code	Meaning
422	`CONSISTENCY_PROOF_INVALID`	A witness cosignature's CC 5.3.1.1 consistency proof against the prior STH it cosigned is absent or does not verify. A missing proof when a prior STH exists, or a tree_size behind the witness's prior cosigned STH, is `MALFORMED_REQUEST` instead.
422	`LINEAGE_HEAD_INCONSISTENT`	A lineage-head cosignature (CC 5.3.1) names a prior head the Registry did not record for that witness, or a head that neither equals nor descends by `prev_head_digest` from it. A quorum-signed head that does not descend from the lineage's *witnessed* head is not this error: it is recorded uncosigned as `hard_case:lineage_fork_attempt` evidence (CC 3.2 T6)
413	`ENVELOPE_TOO_LARGE`	Canonical JCS bytes exceed the CC 2.6.1.3 1 MiB bound. Raised at **every** write path (HTTP body, capsule, FFI, tier-ingest), not only at an HTTP gate — the CC 2.6.1.3 rule is an admission bound, not a transport limit. The storage-layer spelling `federation_envelope_too_large` maps to this wire code; the wire code is the conformance surface.
429	`RATE_LIMITED`	`X-RateLimit-*` headers set; `Retry-After` honored
500	`INTERNAL_ERROR`	Server-side fault; request_id usable for support
503	`WITNESS_DIRECTORY_UNAVAILABLE`	Substrate replication lag exceeds liveness bound

Rate-limit headers on every response: `X-RateLimit-Limit`, `X-RateLimit-Remaining`, `X-RateLimit-Reset` (seconds-until-reset epoch).

5.4 Scope-Native Privacy wire profile

The wire-format absorption of the Scope-Native Privacy construction. These are the deterministic, byte-pinned primitives a conformant substrate uses to make real and cover traffic indistinguishable below the cohort boundary — record addressing, per-symbol AEAD, uniform fragmentation, the MLS Welcome wrap, the witness-chain discipline, and announce suppression. They are additive to the wire layer; the 1+4 surface is unchanged.

5.4.1 Deterministic `record_id` (CBOR profile, normative)

Scope-native distribution addresses each record by an HMAC-opaque `record_id` rather than any plaintext key. The `record_id` is a deterministic, group-keyed commitment to the record's identity: every member of the MLS group derives the identical `record_id` for a given record, while non-members observe an unlinkable 32-byte tag. This is the addressing argument of the deterministic ALM "directory" — a pure function, never a queryable authority (CC 4.4.3.2.1 cohort-scope lattice).

The preimage `record_id_input` is the Core Deterministic Encoding of a fixed four-entry map; `record_id` is its HMAC under the per-epoch key `K_record_id`. The preimage, the key derivation, and the resulting tag MUST be byte-identical across producer, substrate, and every consumer — any disagreement on CBOR byte order, integer minimal-encoding, major-type selection, or HKDF/HMAC suite yields a different `record_id`, silently partitioning holders of the same record. This is the same byte-exactness hazard class the STREAM nonce derivation guards (CC 5.3.3.1).

Per-epoch key. `K_record_id` is a bare HKDF-Expand (no Extract, no MLS KDF-label framing) over the raw group `exporter_secret`:

```
K_record_id = HKDF-SHA256-Expand(
    PRK  = raw group exporter_secret,
    info = "ciris-edge/scope-privacy/record-id/v1"  (ASCII),
    L    = 32
)
```

The HKDF hash here is **SHA-256** (matching openmls ciphersuite 0x004D's key-schedule hash). Implementations MUST pass the raw `exporter_secret` to the `ciris-crypto` `scope_privacy::k_record_id` helper and MUST NOT route this label through openmls's `export_secret()` / RFC 9420 `MLS_Exporter()`.

Preimage and commitment. Each record:

```
record_id_input = CBOR_dCE({
    "v":   1,
    "epc": mls_group_epoch,
    "iid": internal_id,
    "typ": record_type,    // integer per RecordType table below
})
record_id       = HMAC-SHA3-256(K_record_id, record_id_input)
```

`CBOR_dCE` is RFC 8949 §4.2.1 Core Deterministic Encoding (shortest-form integer encoding, definite lengths only, no floating-point unless required), with map keys ordered **by encoded-length first, then lexicographically by encoded key bytes**. The four-entry map at `v=1` therefore serializes keys in the canonical order `v, epc, iid, typ`.

Major types are pinned. Within the map the four keys (`v`, `epc`, `iid`, `typ`) are CBOR text strings (major type 3). The values of `v`, `epc`, and `typ` are unsigned integers (major type 0). The value of `iid` is a **CBOR byte string (major type 2)** — `internal_id` is committed as its raw bytes, never as a text string, even when those bytes are human-readable (e.g. `record-0001`). Encoding `iid` as a text string (major type 3) produces a different preimage and a different `record_id`; the byte-string header is why the vectors below carry `0x4b`/`0x41` (major-2) and not `0x6b`/`0x61` (major-3) at the `iid` value position.

`RecordType` integer encoding (pinned for cross-impl conformance):

Type	int
reserved	0
SelfRecord	1
FamilyRecord	2
CommunityRecord	3
FederationRecord	4

Additional record types extend this table via CEG §11.

Conformance vectors (normative). With `K_record_id = 0x11` repeated 32 times, conforming implementations MUST reproduce these exact CBOR preimage bytes and `record_id` tags (hex):

```
// Vector 1 -- CommunityRecord (typ=3), epc=7, iid="record-0001"
record_id_input = a461760163657063076369696 44b7265636f72642d303030316374797003
record_id       = 5428ddb514a8f8692cc4f254f3550ea75790f5069673e42afb6ef318517a0b21

// Vector 2 -- FederationRecord (typ=4), epc=300, iid="record-0002"
//   (forces 0x19 u16 epc header: bytes[8..11] == 0x19 0x01 0x2c)
record_id_input = a461760163657063190 12c636969644b7265636f72642d303030326374797004
record_id       = 04eebeee4d5b83f2fdd0012a205781e6c05fe9a587377e6161b347629a189ff2

// Vector 3 -- SelfRecord (typ=1), epc=16909060 (0x01020304), iid="x"
//   (forces 0x1a u32 epc header: bytes[8..13] == 0x1a 0x01 0x02 0x03 0x04)
```

```
record_id_input = a4617601636570631a010203046369696441786374797001
record_id       = 79bee8b3f1e815a1df03ca9d83427dc5ab474e184f34e3876d3ef3c36559d6a3
```

The deterministic CBOR profile is normative here, per the CEG §11 record-type registry; the vectors are published with CIRISVerify.

5.4.2 Uniform symbol AEAD framing for A/V symbols (normative)

Within both encrypted crypto tiers — the self/family **InvisibleEncrypted** tier and the community/affiliations **CommunityDek** tier (CC 4.4.3.2.1, resolved through `crypto_tier(cohort_scope, cohort_subkind)` with non-`infrastructure` subkind) — stored content moves and rests as encrypted chunks, replicated per item to its holders and never encoded by any node; the realtime A/V plane (CC 5.3.3.2) alone fountain-codes each sealed chunk into symbols, so a receiver recovers loss without retransmission, and each emitted symbol is sealed under its **own** AEAD key, so a relay or holder lacking the keys holds indistinguishable-from-random ciphertext. The two tiers differ in the shape of their roster, not in discoverability to outsiders, which is none for both: self/family and community/affiliations content alike emit holding claims only within the cohort (CC 5.2 structural invisibility), so outsiders cannot discover the bytes exist; a community's roster may be larger, listed or open (CC 3.2 `public`), a family's is intimate and never listed, and the holder-inspectability principle (CC 4.4.3.2.1) says members hold what they can open and strangers hold nothing (keep/evict decision — but the bytes they hold remain sealed and unreadable: a chunk under its own seal, an A/V symbol under the symbol AEAD below without `K_symbol` and `record_id`). The substrate MUST seal every A/V symbol it emits with the framing below; a holder lacking both `K_symbol` and `record_id` MUST be unable to distinguish a real symbol envelope from a cover symbol envelope.

`record_id` is the per-record HMAC-opaque addressing tag for the CEG §19.3 group-keyed addressing construction — `record_id = HMAC-SHA3-256(K_record_id, CBOR_dCE({v, epc, iid, typ}))`. `K_symbol` and `K_record_id` are the MLS-exporter-derived per-scope subkeys (`HKDF-SHA256-Expand` over the group's raw `exporter_secret`); both rebind on MLS Add/Remove.

For each A/V symbol the wire framing is byte-exact:

```
For each A/V symbol:

    symbol_key = HKDF-SHA3-256(salt=record_id, ikm=K_symbol,
                               info="ciris-edge/scope-privacy/symbol/v1"
                                    || u16_be(symbol_index), len=32)
    symbol_envelope = (record_id, u16_be(symbol_index), nonce,
                       XChaCha20-Poly1305(symbol_key, nonce, fragment), tag)
```

Normative constraints:

- **`symbol_key` derivation is byte-identical across every implementation.** `salt = record_id` (the 32-byte HMAC tag), `ikm = K_symbol` (the 32-byte exporter subkey), and `info` is the **exact** concatenation of the UTF-8 bytes of the ASCII label `ciris-edge/scope-privacy/symbol/v1` followed by `u16_be(symbol_index)` (the symbol index as a 2-byte big-endian unsigned integer), with `len = 32`. The same ASCII label is used both to derive `K_symbol` from the MLS exporter and as this info-prefix; producer, substrate, and every consumer MUST recompute this exact value. Any BE/LE disagreement on `symbol_index`, or any drift in the label bytes, yields a different `symbol_key` → AEAD authentication failure (silent decryption failure).
- **The AEAD is XChaCha20-Poly1305 with a 24-byte `nonce` and the Poly1305 `tag`**

carried in the envelope. The same suite seals **both** real fragments and cover fragments — there is no separate cover cipher, no cover marker, and no length distinguisher. A holder lacking `K_symbol` and `record_id` holds indistinguishable-from-random bytes (Tahoe-LAFS least-authority, by reduction to XChaCha20-Poly1305 IND-CCA + HKDF-SHA3 extract-then-expand); cover and real envelopes are observationally identical.

- **The fragment-set rebinds on MLS Add/Remove.** Because `symbol_key` chains from `K_symbol` and `record_id`, both of which rotate with the group epoch, a removed member's symbol keys do not open post-epoch fragments. Past-epoch `K_record_id` / `K_symbol` are deleted by honest holders per the community's archive policy.
- **`symbol_envelope` field order is fixed: `(record_id, u16_be(symbol_index), nonce, ciphertext, tag)`.** The cleartext `record_id` and `symbol_index` are the addressing argument to the HMAC-opaque directory function (the CEG §19.4 deterministic ALM topology) and are NOT authority-bearing — they reveal nothing to a non-member, who cannot derive the corresponding `symbol_key`.

This framing is the byte-level analogue of the CC 5.3.3.1 per-chunk stream seal: the costly key agreement is amortized into the MLS exporter, and the hot path is a per-symbol symmetric AEAD with a deterministically derived key. It composes with — and never substitutes for — the at-rest DEK cascade (CC 4.4.3.4.1); the symbol AEAD protects the A/V plane's symbols in transit and at any relay that forwards them, the DEK cascade protects the record at rest.

5.4.3 Uniform-envelope fragmentation & reassembly (normative)

A claim's bytes rarely fit one packet, and the split is where privacy is most easily lost: a naive fragmenter leaks payload size, count, and cadence to anyone on the path. The discipline here is that every fragment on the wire — a real publication fragment or a synthetic cover envelope — is the **same size** and the **same AEAD shape**, so an on-path observer cannot distinguish a large publication from idle substrate maintenance. This composes with the chunk-seal of CC 5.3.3.1 (a *different* axis: streaming media chunks) — fragmentation is the substrate-emission layer beneath every Edge `send`, not the streaming codec.

Fixed frame size on a shaped link (normative). On a shaped link (below) every frame is one constant size, the link's channel payload MDU, which differs by interface (about 400 bytes at the Reticulum base MTU, about 16 KB over TCP); a larger payload is fragmented to that size and a smaller one padded to it. Within a link the size is not a tuning knob, because a variable frame re-exposes the size channel that shaping exists to close. An unshaped link carries no padding requirement. There is no fixed envelope size for all outbound traffic: uniform frames add privacy only where both ends shape.

Link shaping (normative). Shaping hides when and how much real traffic a link carries by sending a constant stream of identical frames, real or cover.

1. **A link property, negotiated per link.** Shaping is agreed on each link immediately after the CC 5.3.3.5 E1 handshake, under its keys, not set per node. Each end offers a budget class, `server`, `home`, `phone-wifi`, `phone-cellular` or `off`; the link runs at the lower of the two offers, and a node divides a node-wide cap across its shaped links. Shaping is **off by default** until the steward sets a fleet default.
2. **Both directions or neither.** On a shaped link both directions are shaped at the negotiated rate: a responder that accepts a shaped link shapes its replies on it, and a node

that cannot meet an offer declines the link rather than run it half-shaped. One shaped side beside one unshaped side is exposed by the unshaped side's timed replies.

3. **The stream.** Every frame on a shaped link is a constant-size sealed frame; real frames are fragmented to that size, and cover frames are indistinguishable from them under the seal. Frames leave on a per-link Poisson timer at the negotiated rate λ. Cover dominates by the budget rule: real traffic is at most a fraction ρ of the stream over a window W. ρ and W are explicit settings, because they trade latency against how much load an observer can infer; the slow tail of delivery comes from W, not from the timer. Their defaults are in the table below.
4. **Scope.** Maintenance traffic (anti-entropy, replication, witness, repair) and ordinary rows are shaped. The real-time audio and video plane (CC 5.3.3.2) is never shaped.
5. **What shaping does not hide.** Link setup stays observable: the request to open a link to a derived destination is visible, and shaping runs only inside established links. A node with any unshaped link can be told apart from a node that shapes every link. Shaping hides timing and volume inside a shaped link, not which links exist (CC 5.4.6, CC 1.13.3.1).

Fleet defaults (normative defaults; every value stays a per-node setting). Shaping is **off** by default. When a node turns it on, it offers its budget class with these values:

Budget class	ρ	W	Byte budget per link
`server`	0.5	600 s	7,000 B/s
`home`	0.5	600 s	2,800 B/s
`phone-wifi`	0.5	600 s	1,400 B/s
`phone-cellular`	0.5	3,600 s	350 B/s

A node's shaped links together use at most eight links' worth of its class budget.

Why these values (informative). They come from a model run on the production scheduler and budget meter over a virtual clock with real row sizes. They are not measurements from a live mesh, and they are revised when live measurements exist. What the model found:

- ρ must sit above the link's real load with margin. When real load exceeds ρ the link saturates: useful throughput is capped at $\rho \cdot \lambda$ and the sender is held back. $\rho = 0.25$ halves useful throughput and saturates at ordinary load. $\rho = 0.8$ leaves the peer almost no uncertainty about the real share. 0.5 is the balance.
- Well below ρ, neither ρ nor W changes latency; the Poisson timer sets it (about 1.6 s mean on `server`, 4 s on `home`, 8 s on `phone-wifi`, 26–32 s on `phone-cellular`). Near ρ, W decides: a 60 s window costs one to two minutes at the mean and up to five at the 95th percentile, 600 s costs 6 to 36 s, and 3,600 s returns to the timer's base.
- `phone-cellular` uses W = 3,600 s because at its low rate a 600 s window admits only 75 to 260 real frames, and near the limit delay rose to about 380 s mean and 910 s at the 95th percentile, against 29 s and 54 s at 3,600 s. No phone class uses W = 60 s.
- The cover share is the unused capacity (90 % at 10 % load, 50 % at 50 %), the same at every W.
- The wire learns nothing from a shaped link at any setting. The peer learns the real share, bounded by ρ, and the timing of real frames smeared over W; a relay learns nothing more than the wire.

Uniform AEAD framing across real + cover (normative). Every envelope — real publication AND synthetic cover alike — MUST use uniform **XChaCha20-Poly1305** AEAD framing. A synthetic cover envelope MUST be marked `type=cover` in the **AEAD-protected** header; the `type` discriminator MUST live inside the AEAD envelope so an on-path observer cannot separate real from cover by inspecting cleartext. This XChaCha20-Poly1305 envelope/symbol framing is distinct from the streaming chunk-seal AES-256-GCM of CC 5.3.3.1 and from the HPKE-layer AES-256-GCM of the MLS Welcome wrap; they are different layers and MUST NOT be conflated.

Fragment unit on the A/V plane — the symbol envelope. *(normative; wire-exact)* Only the realtime A/V plane (CC 5.3.3.2) fountain-codes: a sealed A/V chunk is encoded with RaptorQ into symbols so a receiver recovers loss without retransmission, and the code's parameters are the stream's, not substrate constants. Stored content is never encoded by a node: a sealed chunk travels whole, fragmented to a shaped link's frame size as above, and is replicated per item to its holders (CC 6.1.5). Each symbol is sealed independently, with a per-symbol key diversified from the group `K_symbol` (MLS exporter-derived per the CC 5.1 epoch key schedule) and the record's `record_id` (the deterministic-CBOR `record_id` profile):

```
symbol_key = HKDF-SHA3-256(salt=record_id, ikm=K_symbol,
                           info="ciris-edge/scope-privacy/symbol/v1"
                                || u16_be(symbol_index), len=32)
symbol_envelope = (record_id, u16_be(symbol_index), nonce,
                   XChaCha20-Poly1305(symbol_key, nonce, fragment), tag)
```

The per-symbol KDF is **HKDF-SHA3-256** — chosen for **hash-family separation** from the SHA-256 key-schedule hash used one layer up (a domain split between layers, so one family's weakness does not reach both), not for harvest-now-decrypt-later: the HNDL exposure of this plane is the KEM's, which is why the KEM is hybrid (CC 5.1, the hybrid X25519 + ML-KEM-768 construction); a hash choice inside a KDF has no bearing on it. The `info` string `"ciris-edge/scope-privacy/symbol/v1"` (ASCII) concatenated with `u16_be(symbol_index)` is normative and MUST be byte-identical across producer, substrate, and every consumer; a `u16_be` / `u16_le` disagreement on `symbol_index` yields a different `symbol_key` → AEAD verify failure on reassembly. Without `K_symbol` and `record_id`, a holder possesses bytes indistinguishable from random (Tahoe-LAFS least-authority, by reduction to XChaCha20-Poly1305 IND-CCA + HKDF-SHA3 extract-then-expand) — the structural basis for a holder's truthful cold-state-ignorance claim.

Fragment-set rebinds on MLS Add/Remove (normative). `K_symbol` and `K_record_id` are bound to the MLS group epoch (derived from the group `exporter_secret`). On any MLS **Add** or **Remove** the group epoch advances, so the fragment set MUST rebind: `record_id` and `symbol_key` are recomputed under the new epoch, and subsequent symbols seal under the new keying. This carries the forward-secrecy guarantee of CC 5.1 onto the fragment layer — a removed member cannot derive the post-removal fragment set, and a newly-added member receives forward keying without retroactive symbol access except as the archive policy permits (the companion per-community archive-policy section; default `rotate-forward`, 30-day window).

Reassembly (fail-honest). A receiver holding `K_symbol` + `record_id` opens each `symbol_envelope` under its recomputed `symbol_key` (XChaCha20-Poly1305 verify; a failed tag MUST discard that symbol, never accept partial plaintext) and decodes the chunk once $\geq$ `K` valid symbols have arrived; a chunk whose symbols do not reach `K` within its play window is a loss, dropped whole, never a partial decode. Stored content is recovered the other way: a consumer pulls the item's sealed chunks from any holder (CC 5.3.2.1 holder discovery), decrypts them with the cohort key it holds, and verifies the full digest (CC 5.3.2.5). No reachable holder MUST resolve to a `ContentMiss` — an honest miss, never a silent gap or a downgrade to unverified bytes (fail-honest).

5.4.4 MLS Welcome HPKE wrap under invitee static X-Wing key (normative)

MLS Welcome messages MUST be wrapped by **HPKE `mode_base`** (RFC 9180 §5.1.1) encrypted to the invitee's **static X-Wing KEM** public key (ML-KEM-768 + X25519; draft-connolly-cfrg-xwing-kem; confirmed in draft-ietf-hpke-pq §7.2). This is the admission-ceremony half of the MLS construction whose epoch rekey is specified in CC 5.1 — the same hybrid KEM (X25519 + ML-KEM-768) the MLS ciphersuite uses as its HPKE KEM.

`mode_auth` is structurally unavailable. X-Wing has no `AuthEncap` (per draft-connolly-cfrg-xwing-kem; confirmed in draft-ietf-hpke-pq §7.2), so HPKE `mode_auth` cannot be used. Sender authentication therefore MUST NOT be claimed at the HPKE layer; it is provided out-of-band by an **ML-DSA-65 signature from the inviter** over the encapsulation bytes (below). Both halves of MLS ciphersuite `0x004D` are used.

HPKE parameters (cross-impl pinned; private-use suite-id, since X-Wing has no IANA KEM-id):

```
HPKE_SUITE_ID = b"HPKE-xwing-hkdf-sha256-aes256gcm-v1"
KDF           = HKDF-SHA256
AEAD          = AES-256-GCM
```

The `HPKE_SUITE_ID` string MUST be used, byte-for-byte identically on both sides, in every RFC 9180 `LabeledExtract` / `LabeledExpand` call. It is a **domain-separation byte literal, not a wire-vocabulary identifier** (CC 5.1 class rule): its hyphens are KDF input bytes and MUST NOT be snake_cased to match the `wrap_algorithm` vocabulary form. The HPKE-layer AEAD is **AES-256-GCM** — distinct from the XChaCha20-Poly1305 the CEWP substrate uses to protect the A/V plane's symbols and the envelope fragments at its lower layers.

Inviter signature. The inviter MUST sign the encapsulation bytes with ML-DSA-65 over the following length-delimited encoding (no `algorithm` field):

```
encap_signing_bytes
    = x25519_ephemeral_pub      (32 bytes)
    || u32_be(len(mlkem768_ct))
    || mlkem768_ciphertext
```

Signature verification is a **precondition** to unwrapping the Welcome. An implementation that omits it forfeits sender authentication, and the CC 5.2 forwarder-side membership-opacity guarantee no longer holds.

5.4.5 Witness-chain content discipline (privacy witness, normative)

A witness chain that names *what* a peer holds leaks the existence of its records as surely as a holder-discovery attestation does (CC 5.2). The content a witness chain may commit to is therefore scoped by tier, and the federation tier additionally **hides its own rate** so that the *amount* of non-federation activity cannot be read off the chain.

> ***Distinction.*** *(normative; do NOT conflate) This is the* ***privacy witness chain*** *— a content-and-rate discipline on what a peer's witness commitments may reveal. It is NOT the CC 6.1.1* ***wholeness witness*** *(`wholeness_witness:`, the self-published divergence-detection state-root that triggers CC 5.3.2.3 quorum-merge), and it is NOT the CC 5.3.1* ***transparency-log witness*** *(an independent STH cosigner providing append-only / consistency / anti-equivocation guarantees). The three "witness" surfaces are distinct constructions, MUST NOT be cross-verified, and MUST NOT be substituted for one another.*

Federation witness chain — commits to federation-scope `record_ids` + a single rate-smoothed constant-tick counter. A federation witness chain MUST commit only to federation-scope `record_id`s plus a single rate-smoothed counter for all non-federation activity. The counter ticks on a federation-wide **constant** schedule — **1 tick / federation epoch** (CC 5.1) — independent of how much real activity occurred. At each tick the witness commits to a target-relative count, padding below-target ticks with cover leaves and back-pressuring above-target leaves into the next tick:

```
At each federation epoch tick the witness commits to ‘count mod target_rate‘:
real leaves below target are padded by cover leaves to the target;
real leaves above target back-pressure into the next tick.
Cover leaves commit to
‘HMAC-SHA3(witness_signing_key, leaf_position || epoch_id)‘
-- cryptographically indistinguishable from real Merkle roots
under the IND of HMAC-SHA3.
```

Because the tick is constant and the per-tick commitment is `count mod target_rate` smoothed by IND-indistinguishable cover leaves, an observer of the chain learns neither the true non-federation activity rate nor which leaves are real. **The signal is the deviation, not the count**: a witness that fails to tick on the constant schedule has emitted the only observable signal the chain carries, and **deviation from the constant tick is a federation-tier slashing condition**.

Per-community witness chain — member-only visibility, signed inside the group. A per-community (CC 3.2) witness chain MUST commit to the community's `record_id` set with **member-only visibility**, and MUST be **signed inside the community's MLS encryption** (CC 5.1 epoch keying / RFC 9420 TreeKEM). The chain — like the in-group trust graph, attestation, and eviction machinery — runs *inside* the same encryption boundary that delivers outsider opacity: a non-member cannot read the community's `record_id` set off the chain, because the chain never federates beyond the group's encryption. There is no constant-tick rate counter at the community tier; rate-hiding is a federation-tier concern (the federation chain already smooths *all* non-federation activity, of which per-community activity is a part).

This is a normative substrate discipline, composing with the CC 5.2 structural-invisibility promise: structural invisibility keeps self/family content off the discovery surface; the witness-content discipline keeps the *witness* surface from re-leaking, at federation scope, what the discovery surface withholds.

5.4.6 Announce suppression for group-scoped destinations (normative)

A Reticulum **announce** is the broadcast that tells the whole network a destination exists and is reachable. Announcing a *group-scoped* destination therefore leaks the one fact the cohort tiers exist to withhold: that the group exists at all. The discovery discipline closes that leak — group destinations are resolved from a **cached directory + per-group HKDF**, never from an announce. This is the transport-layer realization of CC 5.2 structural invisibility on the *addressing* plane: a bounded cohort's holding claims already never leave it (CC 5.2); here the **destination itself** is suppressed at the announce plane.

No announce for group-scoped destinations (normative). A destination whose `cohort_scope` is below federation (the InvisibleEncrypted and CommunityDek tiers — `self`, `family`, `community`, `affiliations`) MUST NOT emit a Reticulum announce — **directed or broadcast; the prohibition binds the emission, not the addressing mode**. The substrate MUST suppress the announce that would reveal a group-scoped destination's existence. Only federation/Commons-scope destinations (`species`, `biosphere`, `federation`, and the `infrastructure` opt-out) MAY

announce normally — they carry no anonymity claim.

The device roster is public exactly for the devices announced — per node (normative). A person's device roster is public because it has to be for them to be contactable, and it is not thereby *exposed*: it follows this section's lightnet/darknet split **one node at a time**. Each node's setup chooses whether that node **announces**, which in this section means one thing: its owner-binding (CC 3.2) is carried at `cohort_scope: federation`, owner-signed at that audience. The public roster of a person is **exactly the set of their nodes whose owner-binding is federation-scoped** — the devices they chose to be reachable on — and nothing else; the Reticulum announce of a node's transport destination, which every routing node emits, places nothing on the roster by itself. **A node that holds a `serve` seat in an `infrastructure` community MUST have an announced owner:** the point is that a named human is responsible for the node and can be found. While the node holds the seat, a live owner-binding for it MUST exist at `cohort_scope: federation`; it need not be part of the genesis and can be requested at any time. The test is evaluated live, never at a past instant: a binding made after the node was seated satisfies it, and a withdrawn binding stops satisfying it. While no such binding exists the node does not serve under the seat, refused `serve_seat_owner_unannounced`; nothing in the genesis bundle or the holders' ceremony carries or seeds the binding, which is the owner's own row. A node that boots seated and owned but unannounced announces itself if its owner's key is present and otherwise raises an alarm. The check is persist's and the server gates on it. A node whose owner-binding is held only at `self` stays on the derived plane: reachable by the person's own nodes and by whoever holds a v3 code they issued (CC 2.6.8), **never listed**, and a directory, serve policy or read route MUST NOT enumerate it to any node outside the owner's `self` cohort. This mirrors the CC 2.1 `listed` discipline (a roster is never globally enumerable without the member's opt-in) at the identity plane, where re-signing the binding at federation scope *is* the opt-in, and it is the `self` cohort over the owner-binding graph that decides who reaches the unannounced remainder. **Composition with the minors floor:** a federation-scope owner-binding makes its owner contactable and discoverable by unconnected adults, which the CC 3.4.13 Q5 hard floor forbids for a minor and no steward, guardian or stacked consent may lift — so a node whose owner resolves to the `minor` band MUST NOT carry a federation-scope owner-binding, admission MUST refuse the promotion, and such a person is reachable only on the derived plane, through their own nodes and the codes they hand out.

Discovery is directory-cached, not announced (normative). Group members resolve each other's destinations from a locally cached directory plus a deterministic per-group derivation — no announce, no per-resolution query:

```
- Every participating L1 peer maintains a COMPLETE local copy of the
  federation directory's X-Wing public keys, refreshed via the existing
  'federation_keys' anti-entropy. The directory remains federation-public.
- A group destination is resolved DETERMINISTICALLY from
  (cached directory entry + per-group HKDF) -- every member derives the
  same destination; outsiders, lacking the group key, cannot.
- NO per-invitation / per-resolution directory query is emitted.
- Phone-class peers that cannot hold the full directory resolve through
  their L1-relay parent over a relay-blinded path.
```

The derivation (normative). The destination is derived per member, per group epoch, from the group's MLS exporter and the member's `key_id` (the cached directory entry). Its golden vector is published with CIRISVerify:

```
S_dest        = MLS-Exporter("ciris-scope-destination-v1", context = "", L = 32)
k_destination = HKDF-SHA256-Expand(PRK = S_dest,
                   info = "ciris-edge/scope-privacy/destination/v1", L = 32)
destination   = HKDF-SHA3-256(IKM = k_destination, salt = "",
                   info = "ciris-edge/scope-privacy/destination/v1"
```

```
                        || u32_be(len(member_key_id)) || member_key_id,
                L = 16)
```

The expand step is bare `HKDF-SHA256-Expand`, not RFC 9420 `ExpandWithLabel`. `S_dest` is a **separate** exporter from `S_record` (`"ciris-scope-record-v1"`, which feeds `K_record_id` and `K_symbol`, CC 5.4.1). Storage holds the record secret and only transport holds the destination secret, so a compromise of the record layer cannot recompute or link the addresses a node has used. Two members of one group get unrelated addresses. A destination is derived once per `(group, epoch)` at a membership or epoch change, never per packet; it rotates with install-next, activate, seal, so an epoch advance never silences a group.

What an outside observer of link traffic still sees (normative framing). Every node announces its identity destination, so an observer sees that each member node exists and is reachable. For a group it sees link requests to 16-byte derived destinations that it cannot map to a group or a member, the identity of the node that opened each link, and then sealed frames with their timing and sizes. It cannot fetch the content. What this section hides is the group's content, its membership and its address-to-group mapping. It does not hide the fact that node X opens links to some derived destination, or the shape of that traffic, which is the relationship-existence and traffic-analysis boundary of CC 5.2 and CC 1.13.3.1.

The anonymity target is outsiders, not in-group members (normative framing). Trust-enforceability to insiders and anonymity to outsiders are the same property viewed from opposite sides of the cohort encryption boundary (CC 1.13.3.4). In-group members already hold the per-group key and resolve every peer's destination from the cache — they see the group fully. The construction hides the group's existence, membership, and `querier` → `invitee` edges from **outsiders only**, including a subpoena against the federation-public directory: because no announce is broadcast and no resolution query is emitted, the directory exposes no edge linking a resolver to the resolved destination. *Outsiders* here means the directory and any reader of wire identifiers; against an on-path forwarder's observation of link traffic the claim is CC 1.13.3.1's, not this section's.

Fail-secure. A peer with a stale or incomplete directory cache resolves *fewer* destinations and reports an honest miss; it MUST NOT fall back to emitting a group-scoped announce or a per-destination query to recover reachability. Per-destination announce control is a small Leviculum substrate extension (informative); its absence MUST fail toward suppression, never toward announce.

A directed announce inherits the prohibition (normative). A targeted announce inherits the prohibition rather than escaping it. The purposive sentence above — suppress *the announce that would reveal* — is the prohibition's rationale, not a directed-announce exception, because on Reticulum transport no directed announce can satisfy it: multi-hop path learning **is** outsider observation — non-member transport nodes must observe the announce and retain path state for the destination, route entries a subpoena then reaches — and the epoch-bound derivation (above) forces a dilemma: a derived destination that rotates with the MLS epoch emits a synchronized roster-wide re-announce wave on every Add/Remove (cardinality, timing, membership churn — the group-existence facts this section withholds), and one that does not rotate leaves a removed member holding every peer's addressing indefinitely, breaking the CC 5.4.3 rebind discipline on the addressing plane. The fail-secure rule bans the targeted, non-broadcast per-destination *query* in the same breath: the object is the emission class, not the broadcast. What a directed announce offers is a trade of *no emission exists* (structural, honestly claimable) for *emissions exist but resist analysis* — traffic-analysis privacy, which remains a live goal and is precisely the claim base CEG/RET declines to make (CC 1.13.3.1; the Anonymous Tier is its opt-in). A reading

MUST NOT spend a claimable guarantee to purchase an unclaimable one. An emission that never leaves the group's encryption reveals nothing to outsiders, and that object already exists here as the cached-directory discipline; in-group distribution of addressing material over MLS is not an announce and is not prohibited. Multi-hop reach for a group-scoped destination is therefore an **amendment-plane design question** — its bar: no outsider-observable emission, no outsider-retained path state, no epoch-correlated wave — never a reading of this clause; announce-free reach (direct link; relay-blinded resolution) is the sanctioned posture.

A scoped body MAY ride the identity-plane link when no direct path exists, and the intermediary's only role is reachability (normative). The derived address above is the zero-observer path: where a direct link exists between two members, a scoped body crosses it and nothing else. Where the only path between two members runs through a non-member transport node — two phones behind NATs with no member node between them — a scoped body MAY ride the members' **identity-plane link**: end-to-end encrypted between the two endpoints, the room discriminated *inside* the link encryption, and admitted by the receiver against the same address table the derived-address arrival path consults. Nothing about the derived address is emitted, resolved or retained by anyone; this rule changes what a link *carries*, never what is announced, and the prohibition above is untouched. What the forwarder learns is exactly what CC 1.13.3.1 already concedes at federation scope — that a link exists between two announced endpoints and that ciphertext flows on it at certain times and sizes — and nothing more: it cannot read the body, cannot tell a room body from a replication round, and holds no identifier that names the room. **The intermediary's sanctioned role is reachability, never custody of the conversation.** A non-member transport node exists to help two members reach each other: it answers the path request and carries link establishment. Where the endpoints can bring the link direct — a shared interface, or a direct dial on an address the path exchange revealed — the link MUST be brought direct and the forwarder drops out; where they cannot, the forwarder carries link ciphertext only, retains nothing beyond the transport table the substrate keeps for every link it forwards, and MUST NOT be handed anything that names the room. A substrate MUST prefer the derived address whenever a direct link exists and MUST NOT route a scoped body through a forwarder when a direct path is available. What is conceded is precisely the CC 1.13.3.1 non-goal at the scope where it is already conceded: an on-path observer correlating link traffic across a clique of members can infer that a group exists, which is traffic analysis; the record layer's diversified record ids and cover framing (CC 5.4.2, CC 5.4.3) are the wire-privacy answer for the *payload* and are not a claim against that inference. The mechanism that would hide the endpoint correlation from the forwarder as well — blinded, rotating retained state, the Tor v3 / I2P b33 family the Position below names — is the more secure option and stays on the roadmap; a deployment that needs it layers it. Minors are unaffected: reachability still requires the announce a minor's node cannot carry (the device-roster rule above).

The Anonymous Tier: onion routing (normative). The Anonymous Tier (CC 1.13.3.1) hides who published a federation-scope item from the nodes that store it, from the directory, and, with shaping, from an observer of the network.

1. **Using it is opt-in; relaying for others is the default.** A publisher chooses to publish through an onion route. Every node relays onion traffic for others by default, because a relay set made only of volunteers would make its operators targets, and a node that runs the standard substrate is a relay like every other. A node that holds no serve seat MAY opt out of relaying by publishing the opt-out marker of CC 3.3.6.2 and withdrawing its hop key on the same superseding row; a serve node MAY NOT: a seat carrying the marker is out of conformance as `seat_opted_out`, and a seat without a valid hop key as `seat_without_hop_key`. A node with no hop key and no marker is not yet capable of relaying and is

simply not placed. **Device policy.** A phone pays for relaying in data and in a radio that never sleeps, so a phone-class node relays by default only on Wi-Fi while charging, and not on battery or on a metered connection; the user MAY override the policy either way, and the client states the cost of each setting. Servers and home nodes relay by default without qualification.

2. **First scope.** The tier carries federation-scope publication. Other scopes are not routed through it.

3. **The packet.** An onion packet is a layered packet of fixed length in which each hop's layer is encapsulated to that hop's X-Wing onion hop key (CC 3.3.6.2) and protected by a key derived from that encapsulation; each hop removes one layer and learns only the next hop. Every layer is post-quantum: a construction whose per-hop key agreement is classical alone is not conformant (CC 5.3.3.5 E1). Because ML-KEM cannot be blinded, the header carries one KEM ciphertext per hop.

4. **The route and the layers.** A route has exactly three hops, one from each of three layers. **Who is a relay for a period:** a node whose current signed `transport_destination` row is live and announced, carries a valid hop key and no opt-out marker, and whose identity is owner-bound under a trust root in the genesis bundle's charter set (the bundle is the same everywhere, so every node reaches the same answer, and an unowned fresh device is never a relay); at use, a relay's layer links MUST be shaped, and an unshaped link carries no onion packet. **The deal, once per rendezvous period on the global clock:** the relays are sorted by `H(key_id ‖ period)`, serve seats first and then the others, and dealt round-robin across the three layers, seats first. Rotation by period gives a relay held back by the cap its turn. **The cap:** in each layer the other relays number at most $\lceil c \times$ conferred relays in that layer$\rceil$, with `c` a steward setting (default 2), where a conferred relay is a serve seat or an identity that holds the steward-conferred `infra:relay` role (owner-bound, rooted under a bundle root): the bound stays where trust is conferred, so no layer is ever more than two-thirds unconferred relays and a swarm of strangers cannot take a layer, and the steward grows the layers by conferring; relays past the cap wait for the next period's deal. **The floor:** a layer with fewer than three members makes the tier unavailable for that period; the tier never runs a thin layer. **Availability** is two questions. The deal answers *who*: identically on every node, from the fold. A per-node liveness measurement answers *whether now*: a relay counts as answering when it completed a shaped handshake, answered a route request, forwarded a packet the next hop acknowledged, or returned a self-addressed probe within the period; the tier is available at a node when the floor holds on the deal (else `onion_tier_unavailable_floor`) and at least three relays per layer are answering (else `onion_tier_unavailable_liveness`), and the client shows those counts, which may differ between nodes and are the honest measure of the crowd (below ten per layer the client says the set is small). Every rule here is a pure function of the fold, never of arrival order, so no coordinator assigns the layers and two nodes holding the same rows deal the same layers.

5. **The carrier.** Onion packets travel as the bodies of shaped frames (CC 5.4.3), so a link carrying onion traffic looks like any other shaped link. Each hop is an ordinary sealed, attributed link between relays: the storage node sees the last relay, never the publisher.

6. **Rendezvous.** A receiver that accepts targeted anonymous delivery publishes the route to itself as a `rendezvous:route:v1` row (CC 3.1.4), signed by a fresh rendezvous key each hour, with the route reused for every message in that hour. The rendezvous node that holds the receiver's mailbox MUST be a serve seat, because seats do not churn and an hour-long route through a node that sleeps is lost; one seat holds each mailbox. A sender builds its own three-hop header from the directory's hop keys and appends the receiver's route.

7. **Rotation.** Hop keys, layer assignments and rendezvous indexes rotate on global clocks, never on group events.
8. **What it does not yet buy.** A relay can count the packets sent to one rendezvous in a period, because the route is reused within it. Mixing hides a publisher only among the traffic a relay carries in a window; with few users per relay the anonymity set is small, and user-facing copy says so (CC 1.13.3.4). Shaping is a prerequisite: without it the use of the tier is visible even where the path is not.

Position. *(informative; prior-art record)* Against the field, this design sits in the zero-emission corner of the anonymity trilemma — the membership-concealment corner, where reachability information flows only to authorized parties — with the impossibility floor ("whoever relays for you must know how to reach you"; Vasserman et al., CCS 2009) satisfied entirely by members. The stance it realizes is CC 1.13.3.4: anonymity at the smallest cohort scope by default, federation scope as the opt-in, claims bounded by CC 1.13.3.1. What is distinctive is that the corner's two known prices are bought back rather than paid. First, the **two-plane split** — the **lightnet / darknet split**: the identity plane is public (**lightnet** — announced, rooted, attributable; the federation directory announces normally and makes no anonymity claim), while the group plane (**darknet**) is *derived* from it (cached directory + per-group HKDF, above) — derivation replaces discovery, so no emission exists to observe and the classical darknet bootstrap problem dissolves; surveyed systems that hide group existence otherwise accept out-of-band address exchange. Second, **determinism replaces coordination** (CC 6.1.6): the member-relay ALM tree is a pure function every member computes identically, recovering $\lceil \log_k N \rceil$ fan-out without the coordinator that would otherwise have to learn the group exists — the surveyed zero-emission systems accept linear fan-out as that corner's price, and the systems that kept coordinator-efficient fan-out uniformly declare group metadata out of scope (RFC 9750 delegates it to this very layer). One corollary is recorded without minting a conformance surface: N→1 deterministic aggregation under the CC 6.1.2.1 dominance gate composes with the MLS-epoch roster, steward-binding (CC 6.1.8 N1), quorum-above-time ordering (CC 5.3 `MergeBallot`), and the CC 5.4.5 member-only witness chain into a ballot process whose *occurrence* is invisible to outsiders at scopes below federation — a classical secret ballot hides the vote; structural invisibility hides the election. For any future multi-hop amendment, the blinded-retained-state family (Tor v3 / I2P b33) is the nearest admissible relaxation under the bar above, with the field's rotation rule made explicit: rotation clocks must be global, never group-event-driven.