Sovereign Personal Agent (SPA) - Architecture

A design document. This specifies an architecture for individual data sovereignty: a lifelong, portable personal substrate an individual controls across platforms and over time, with every disclosure governed by scoped, revocable, audited grants. One of its sharpest applications is a Sovereign Personal Agent - a personal agent that acts for the individual, not a corporation, government, or platform - and it is the application this document is framed around. The contextualization underneath is more general than the personal case: the same substrate-lens-frame governs any substrate, not only an individual's data (an enterprise data fabric, or a model's external memory), with individual sovereignty as the flagship and the mission. It is a design, not a built system. The protocol core it rests on, Substrate-Lens-Frame (SLF), is implemented and conformance-tested in slf-core; the broader agent described here is specified, with the recovery path existing as a private prototype. Read it for where SLF is headed, not for what runs today.

> **Mission.** Create autonomy and sovereignty for individuals - not oligarchs, governments, or corporations. Where mission and engineering preference conflict, mission wins.

Companion to the position paper *The Governance Gap in Agentic Memory* (Crenshaw, 2026) and the repo's SLF architecture and threat model.

Author: Andrew Crenshaw (ORCID 0009-0006-6459-0187), Lexenne.
Version: 0.1 (public design draft), 2026.

---

0. Executive summary

The mission is to create autonomy and sovereignty for individuals - not oligarchs, governments, or corporations. Every architectural decision in this document derives from that mission. Where mission and engineering preference conflict, mission wins.

A personal agent that acts on behalf of an individual - not a corporation, government, or platform - is one of the architecture's sharpest applications, and the one this document specifies: persistent lifelong memory, sovereign identity, and an open bridge to the institutions of the world. The same contextualization layer governs other substrates too; individual sovereignty is the flagship and the mission.

The thesis: the components needed for this to exist are now mature (cryptographic identity, verifiable credentials, local-capable LLMs, federated storage, hardware-backed key custody), and the EU has legally mandated the identity layer be in place by end of 2026. **What is missing is the connective layer: a way for a swappable agent to act on a sovereign substrate under the user's control, and for that substrate to interoperate with the institutions of the world.** Two connections do this work - a grant plane between the agent and the substrate (Layer 2), and a bridge protocol between the substrate and the outside world (Layer 4) - and they are what this architecture specifies.

The architecture has **two primitives** entangled in every prior attempt:

- **Primitive A - the sovereign substrate.** Identity, profile, vault. Owned by the individual, portable across hosts, queryable under explicit grants.
- **Primitive B - the personal agent runtime.** Reasoning, action, persistent memory, human-in-the-loop (HITL) gates. A consumer of A, swappable at any time.

Prior attempts built one or the other. Solid built A without B and is a fridge with nothing in it. Personal.ai / Rewind / Limitless built B without A and are corporate-owned "personal" agents. **What every prior attempt left out is the connective layer between them - the grant plane that lets an agent act on the substrate, and the bridge protocol that carries the substrate out to the world.**

The protocol has **one operational primitive** that recurs at every layer: **Substrate / Lens / Frame** ("SLF"). Every credential issuance, presentation, grant, action, attestation, lens query, and audit event is encoded as an SLF object. The primitive is defined once (§3) and inherited everywhere.

Three things must be cleanly separated to make sovereignty actually work for the general public:

Component	Sovereign?	Where it lives	Default for general public
Keys (identity)	Always	User's device - never leaves	Phone secure enclave (passkey-style)
Vault (semantic state)	In control	User's choice - local or encrypted-at-host provider	User-chosen vault provider, ciphertext only
Agent (runtime)	No - fully fungible	Anywhere, anytime	Mix of on-device + user-chosen frontier

This is the SMTP/IMAP architecture applied to personal data. Email proved this works at planet scale with providers competing on UX/price/privacy and users moving freely between them. We apply the same shape to the personal substrate.

One orienting note before the detail. Most of what follows composes pieces that already exist or are mandated: cryptographic identity, verifiable credentials, the substrate, the lens, the grant model, the agent runtime. The genuinely new connective piece - and the destination of this document - is the **bridge protocol (Layer 4, §8)**, the wire-and-trust layer that carries a fact between a sovereign substrate and the institutions of the world. The sections before it build the frame of reference; §8 is what they build toward. Read with Layer 4 in mind.

---

1. Principles

These are the load-bearing constraints. Every design decision below derives from these. When in doubt, return here.

1. **Person-granularity over corporation-granularity.** The system answers to people, identifies people, makes claims about people. Affiliations with corporations are metadata attached to people, never the other way around. *(A unified Person-entity pattern.)*
2. **The DID (decentralized identifier) is permanent. Everything else is a pointer.** Identity is the only piece that cannot be migrated, replaced, or hosted. All other components point to it. This is what makes the rest of the architecture fungible.

3. **Semantic state at ingestion, ontology lens at retrieval.** Durable facts are preserved without freezing interpretation. Ontology categories (health, financial, learning, social, chronicle/timeline) are *lenses* applied to facts at query time, never schemas imposed at ingestion. *(Lifted from Ashwin Gopinath / Sentra, "Are We Learning the Wrong Bitter Lesson?")*
4. **Append-only with provenance.** Facts are signed observations from named sources with timestamps. Retractions are superseding signed facts, not deletions. *(Implemented in a reference substrate's bi-temporal model.)*
5. **Fungibility - no component is load-bearing for any other.** Vault provider, agent runtime, device, network, jurisdiction, frontier model - each is swappable without affecting the others. Lock-in is rejected by design at every layer.
6. **Local-first by default, hosted by choice.** Sovereignty does not require self-hosting. It requires that hosting is a user-revocable choice, not a precondition.
7. **Encrypted-at-host.** Vault providers store ciphertext, not plaintext. The provider cannot read what it holds. Migration between providers does not expose data.
8. **Agents act under grants, not under impersonation.** Agents hold capability tokens issued by the user's DID, scoped, time-bounded, revocable. No agent ever holds the user's signing key.
9. **HITL for critical decisions, autonomous for the rest.** The user defines what is critical. Defaults are conservative; the user expands the autonomous envelope over time.
10. **Standards alignment over standards invention.** Where W3C, IETF, ISO, OpenID Foundation, or EU regulators have specified a wire format, we adopt it. The protocol's value is the *wiring*, not the *invention*.
11. **Layered policy with monotonic restriction. Gates are intrinsic to the substrate, not layered above it.** A fact carries its own constraints (HIPAA, GDPR, audience restriction, expiry, no-further-disclosure) as signed metadata that travels with the fact through every operation. **Lens** is the per-context projection (user identity, current role, current jurisdiction) - what *this user, in this role, in this jurisdiction* sees through the substrate. **Frame** is the run-time outcome - *what is being done right now* within the lensed view. Restriction flows substrate → lens → frame; each layer can only

narrow what the previous allowed. The agent never reasons about regulation; it queries through a lens, and a conforming engine enforces the substrate gates before the lens sees them - structurally where the user holds the substrate, and by receipts and accountability behind a counterparty (§6.3, §7).

12. **The user is always the source of truth.** When third parties hold copies of user data under Copy / Sync / Write-back grants, those copies are working copies, not new substrates. The user's personal substrate is canonical. All changes outside the user's direct supervision must be ingestible back into the substrate with provenance. Where possible, updates flow back to third-party copies. Revocation ends ongoing use; existing copies become stale but cannot be reached by the protocol.
13. **Minimalism is survival.** The protocol does the minimum that makes interoperable, person-mediated exchange possible. Everything else - vault implementation, lens execution, agent runtime, governance of trust lists, vocabulary curation beyond a tiny core, regulatory interpretation, app-level UX - is delegated to other layers, other registries, or other actors. If the protocol is in the critical path of every government regulation and business use case on the planet, it will be forked or bypassed. The smaller the protocol's surface, the less there is to fork over.
14. **Audit, not automate.** The protocol's job is to surface what happened and what is being asked, with cryptographic evidence. It never decides for the user. The human (or institution acting through an authorized agent under a scoped, audited grant) retains decision authority. Every protocol operation produces a signed audit receipt as a first-class output (§3.6); receipts are how trust is earned and how attacks are detected.
15. **Substrate is for facts that change over time. Static attestable attributes can be re-proven on demand from their authoritative source.** The vault holds the durable, mutable, person-shaped record (vaccinations, employer, insurance policies and warranties, qualifications-as-they-evolve). One-shot static government attributes (passport authenticity, citizenship, date of birth derived from passport) do not need to be persisted in substrate; they can be re-proven by the user from the underlying authoritative document each time, with scope-isolated nullifiers. This is the layer that adjacent protocols (e.g., Celo's Self Protocol) occupy; we compose with them rather than replicating them.

16. **No single vendor owns the protocol.** The spec, vocabulary registries, conformance suite, and reference implementation are open and permissively licensed, so the protocol is adoptable without its author. A neutral steward holds the registries and marks - the maintainers today, on the IANA and standards-body model, with a dedicated mission-locked nonprofit reserved for when scale or funding warrants it rather than committed up front (§13). Commercial implementations (including Lexenne's own) operate under the same conformance discipline, with no privileged access.
17. **The protocol is not responsible for misuse.** SMTP is not responsible for email scams; HTTP is not responsible for phishing; SLF is not responsible for what a vault provider, lens engine, agent runtime, or app does with the primitives the protocol provides. The protocol provides the smallest set of primitives that make sovereignty operable; the layers above carry their own accountability. Misuse is addressed by the trust-tier and rating-system mechanisms (§7, §11), not by inflating the protocol's responsibility.

---

2. The two primitives

Primitive A - Sovereign Substrate

The substrate the individual owns. Three components:

- **Identity** (Layer 0) - a lifelong DID with hardware-backed key custody
- **Personal Data Plane** (Layer 1) - federated data: source registry + semantic state + lens registry
- **Capability & Grant Plane** (Layer 2) - user-issued capability tokens authorizing reads, writes, storage, sync, and actions

Primitive B - Personal Agent Runtime

The reasoning system that acts on A. Three components:

- **Reasoning** (Layer 3a) - LLM inference (on-device, local, or cloud - user's choice)
- **Action executors** (Layer 3b) - the components that turn an authorized frame into a real-world effect under a capability token: a scheduler that

books an appointment, a form-filler that submits a benefits enrollment or a loan application, a purchasing agent that places a reorder, a presenter that hands a verifier a signed credential, a sync worker that writes provider results back. Each acts only within its grant.

- **Decision traces + HITL** (Layer 3c) - audit and human approval surface

The Bridge Protocol (Layer 4)

The wire format and trust model that connects the sovereign substrate (A) to the outside world - institutions, providers, other people. (The agent runtime, B, reaches the substrate through the grant plane in Layer 2, not through Layer 4.) As the diagram shows, Layer 4 sits beneath both primitives and bridges the substrate outward.

---

3. The SLF primitive

The protocol has one uniform shape across all operations: a **Substrate / Lens / Frame** ("SLF") object. Every credential issuance, presentation, grant, action, attestation, lens query, and audit event is encoded as an SLF.

This is the operational primitive that makes the architecture coherent. Without it, substrate semantics get redefined three times (in the data plane, the grant plane, and the agent runtime), each with subtly different invariants. With it, substrate/lens/frame is defined once and inherited everywhere.

3.1 The three components

Substrate - the immutable, addressable, self-constraining thing being operated on or referenced. Carries its own type, provenance, and intrinsic gates. The substrate exists independently of any particular operation; operations refer to it.

Examples: a set of signed facts; a personal DID and its claims; a capability token issued by a user; an asset record with full provenance.

Concretely, the substrate is a governed data model, not a particular database. It can be a purpose-built signed-fact store, or the same model layered over a system you already run - a SQLite file, a data warehouse, a CRM (customer relationship management system) - without moving the data.

Lens - the context-bound projection through which the substrate is being viewed. A lens binds a contextual role and jurisdiction to the substrate at a moment of use, filtering and shaping what is visible. Lenses respect substrate gates by construction; they cannot bypass them.

Examples: take one employment record, written to the substrate once. Your own Career lens shows the full history and the salary; a prospective employer's Verifier lens sees only "held this title, these dates"; a lender's lens sees only "currently employed: yes." One set of facts, three lenses - each a narrower projection than the last, and none able to reveal what the facts' gates withhold.

Frame - the run-time outcome being pursued. Identifies the actor, the intended outcome, the moment, any required approvals, and the audit context. A frame is the unit of authorization: grants authorize specific frames; HITL approval happens at frame boundaries.

Examples: "schedule appointment" (within Patient lens); "submit application" (within Job Seeker lens); "consent to records release" (within Patient lens, requires HITL).

3.2 Why this is a first-class primitive

The same triadic shape recurs everywhere in the architecture:

Use	Substrate	Lens	Frame
Data retrieval	Facts with intrinsic gates	Categorical projection (Health, Career, Chronicle/ Timeline…)	The query's run-time context
Identity projection	The Person/Self record	Typed view (Patient, Employee, Family Member)	The current role being inhabited
Authorization	The grantable resource (data, action, capability)	Per-context view (who/where/under-what-role)	The specific action being authorized
Credential issuance	The facts being asserted	The issuer's view of the subject	The issuance event
Outbound assertion	The facts being shared	The receiver's view (what they're allowed to see)	The presentation event
Audit			The audit's purpose

Use	Substrate	Lens	Frame
	The historical operation	The reviewer's role and authority	

Promoting SLF to a first-class primitive instead of three coincidentally-similar implementations buys five concrete things:

1. Three layers shrink to one primitive - L1, L2, L3 stop redefining substrate semantics
2. New operation types compose for free - every new operation is an SLF, not a special case
3. Audit becomes self-documenting - every SLF carries enough to be replayed or analyzed
4. The architecture doc and the wire format share vocabulary down to the type level
5. Teachable in one sentence - "every operation is substrate + lens + frame"

3.3 The SLF object (working sketch)

This is a sketch, not the wire-format spec. The intent is to show shape, not commit to syntax.

```
SLF object:
  version: "1.0"
  type: "credential_issuance" | "presentation" | "grant" | "action" |
        "attestation" | "lens_query" | "audit_event" | ...

  substrate:
    @type: "FactSet" | "IdentityRef" | "CapabilityRef" | "AssetRef" | ...
    ref: <DID or content-addressed identifier>
    gates: [<intrinsic gate metadata, signed>]
    provenance: {...}

  lens:
    @type: <identifier registered in the lens registry>
    subject: <DID of the user under whose authority this operates>
    role: <contextual role this lens binds to>
    jurisdiction: <jurisdictional context for substrate-gate evaluation>
    projection: <lens-specific projection parameters>

  frame:
    outcome: <what is being attempted>
    actor: <DID of the acting party, often an agent>
    moment: <timestamp>
    approvals_required: [<list of HITL gates that must fire>]
    audit: {...}

  payload: <operation-specific data>

  signature: <signature over the entire SLF, by the actor's DID>
```

Every protocol operation is one of these. The evaluation chain (substrate-gates → lens → frame → HITL approval) is a single general algorithm operating on this primitive. In functional terms it is `render(substrate, lens, frame) -> receipt`: evaluating an SLF applies the lens projection and frame check under monotonic narrowing and emits a signed Receipt SLF (§3.6) as its first-class output, returning the gate-filtered view to the caller alongside it.

3.4 How layers use SLF

- **Layer 1 (Data Plane)** - queries, mirrors, and federations are SLFs with substrate = facts and frame = the use context. Lens registry maps lens identifiers to query functions.
- **Layer 2 (Grant Plane)** - grants are SLFs with substrate = the grantable thing and frame = the authorized outcomes. Issuance, revocation, use, and audit are all SLFs.
- **Layer 3 (Agent Runtime)** - agent actions are SLFs with substrate = the facts/capabilities being used and frame = the outcome being pursued. Decision traces are SLFs about prior SLFs.
- **Layer 4 (Bridge Protocol)** - inbound credentials and outbound assertions are SLFs crossing the trust boundary, with EUDIW-aligned wire format underneath.

3.5 The name

SLF is the protocol's name, written as the initialism and pronounced as letters (S-L-F). Like JWT or TLS, one name covers the protocol and the things it carries: the SLF protocol, an SLF object, an SLF grant, an SLF receipt. Layers are named descriptively under it (the SLF bridge protocol, the SLF discovery document), with no separate proper noun. It is not pronounced "self" - the Celo Self Protocol (§12) holds that spoken slot, and we integrate with it rather than compete.

3.6 Audit receipts as first-class output

Every protocol operation produces a signed **Receipt SLF** as its first-class output. This is not a side-channel or an optional log; it is the operation's audited result, returned to both parties and persisted in the user's substrate.

A Receipt is itself an SLF (the architecture describes itself in its own terms - recursive by design). Its shape:

```
Receipt SLF:
  version: "1.0"
  type: "audit_receipt"

  substrate:
    @type: "OperationRef"
    ref:  <SLF id of the operation being audited>
    operation_type: "presentation" | "issuance" | "grant" | "revocation" | ...

  lens:
    @type: "audit"
    subject: <DID of the user>
    role: <auditor role: "operating party" | "regulator" | "user-self">

  frame:
    actor:    <DID of party who initiated the operation>
    audience: <DID of receiving party>
    moment:   <ISO-8601 timestamp>

  payload:
    outcome: "granted" | "denied" | "partial" | "error"
    reason_code: { vocabulary: <URI>, code: <token> }   # see §7.7 abstraction rule
    reason_detail: <free-text>
    disclosed_fields: [<substrate field paths rendered>]
    redacted_fields:  [{path: <...>, reason: <code>}]
    gates_evaluated:  [{gate_id, vocab_version, result, reason}]
    grant_ref:        <SLF id of the grant invoked, if any>

  signatures:
    actor_signature:           <signature by actor DID>
    audience_countersignature: <see §8.7 for v1 counter-signed vs v2 ZK-attestation
modes>
```

Why this is load-bearing. Receipts are the audit spine that makes the trust model verifiable. They are what: - A user inspects to see what was disclosed to whom under which gate - A regulator subpoenas to see whether a provider honored the constraints - The conformance suite (§11.6), once built, tests lens engines against - The rating system (§8.5) feeds on to score counterparty behavior - Detects social-engineering attacks at the standing-grant UX layer (§8.9) - pattern detection across receipts surfaces anomalous standing-grant exfiltration before headline damage

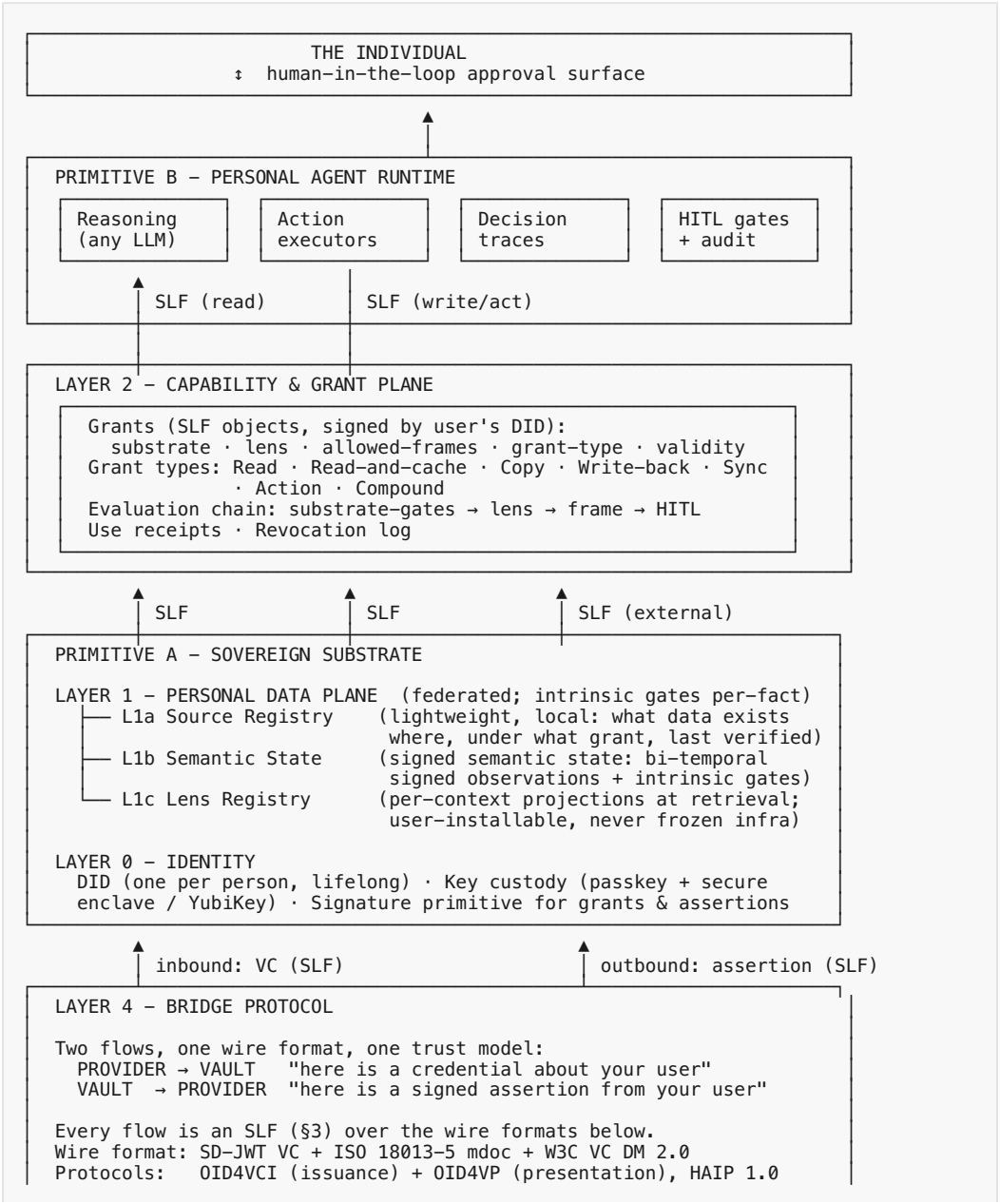
Two receipt modes, provider-electable (detailed in §8.7). v1 counter-signed receipts give maximum cryptographic non-repudiation but create legal-discoverability liability for counter-parties; v2 attestation with a zero-knowledge (ZK) proof gives mutual proof of evaluation without bilateral retention of disclosure detail. Receipts are required; *which mode* is a deployment choice negotiated between parties via the discovery doc.

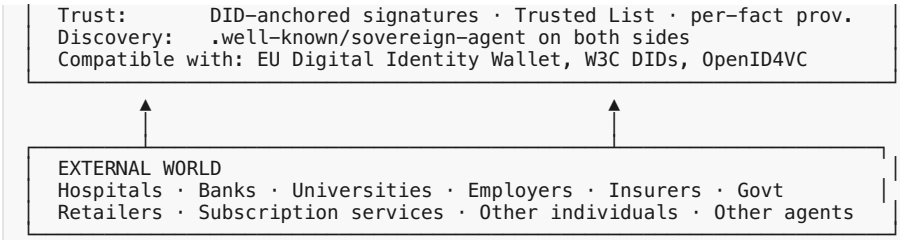
Receipts never carry the disclosed data itself. Only field-path references and gate evaluations. The data lives in the substrate; receipts describe what crossed the bridge, not what was crossed.

Consent-grant content reuses existing standards. Where a receipt or gate references a consent grant - the purpose, lawful basis, controller, recipients, and retention a user consented to - it profiles the ISO/IEC TS 27560 consent-record structure and the W3C Data Privacy Vocabulary rather than minting a parallel format (Principle #10, §12). What is new here is the signed, hash-chained, payload-free *per-operation* receipt on top of that record; 27560 standardizes the grant-time consent record, not the runtime audit spine.

---

4. Architecture diagram





5. The three deployments

5.1 Keys

Default: Phone secure enclave, passkey-style. The phone is already where most users keep their identity (Apple Wallet, banking apps, 2FA). It is also the device they protect most carefully.

Paranoid mode: Hardware key (YubiKey, etc.) as the sole signing root, with phone as a delegate held by the hardware key.

Recovery. The hard problem in personal sovereignty, and the one this architecture prototypes rather than defers. The approach is threshold signing (FROST), not key escrow: the signing key is split into shares - a 2-of-3 default across the device, an encrypted cloud share, and a recovery code, with a 3-of-5 high-assurance tier that adds institutional guardians - and the full key is never reassembled in one place. Recovery rebinds a new device from any authorized pair of shares, behind a cooldown-then-probation window gated by an identity-anchor proof, and every step emits a receipt. An initial reference implementation demonstrates the load-bearing parts: 2-of-3 FROST signing in which the key is provably never reconstructed in memory, the recovery state machine (a 48-hour cooldown and a 24-hour probation, cancellable), and a receipt chain that audits a recovery end to end - each checked by mechanical tests. What is built is the protocol core; what remains is deployment and the long tail of UX: binding shares to platform hardware (iOS Secure Enclave, Android StrongBox) and real cloud-share custody, a publicly-verifiable proof-of-backup, the higher tiers and their cooperative-guardian governance, and inheritance and incapacity flows. One limit is irreducible and stated plainly: if a user loses every share at once with no anchor reachable, the accumulated substrate is gone - re-provable static credentials (Principle #15) survive, the substrate does not. Recovery is designed and prototyped at its core, with consumer-grade UX and platform deployment as the work before general-public release, not a blank space.

Non-negotiable: The signing key never leaves the user's device(s). Vault providers, agent runtimes, network operators - none of these ever hold key material in clear. All signatures originate on user hardware.

5.2 Vault

Three deployment modes the user can choose between (and switch between):

1. **Self-hosted** - user runs their own server (network-attached storage, a home server, or other personal hardware). Maximum sovereignty, requires technical comfort.
2. **User-chosen hosted provider** - provider stores encrypted blobs, runs encrypted queries, has no plaintext access. Provider market competes on UX, price, privacy reputation, geography, integration. *Default for general public.*
3. **Distributed / hybrid** - sensitive slices on the phone, durable mirrors at a provider, federated queries to sources of record. *A mature, distributed default.*

Encryption discipline. The model is client-side encryption: the vault is encrypted on the user's own device, under keys derived from the user's DID, *before* anything is uploaded - so a conforming provider receives and stores only ciphertext, and cannot read what it holds. Migration between providers is a re-upload of that ciphertext plus a pointer update; the provider never sees plaintext. The honest limit is the one that also governs gate enforcement (§7): the protocol cannot force a remote provider to behave. What gives a non-self-hosting user certainty is that the *client* does the encrypting - a conforming client hands over ciphertext only, so however the provider behaves at rest, there is no plaintext for it to expose. A host that keeps plaintext, or that encrypts only in transit, is simply not conformant. The user checks the model three ways: the provider declares it in its discovery document (§8.4); an attested runtime can prove the provider runs the conforming software; and the rating system plus receipts surface a provider that deviates. Absent client-side encryption you are trusting the provider's word, which is the weaker posture - the protocol names it rather than papering over it.

The vault provider market. The hosted mode is a market, not a single product. The same encrypted vault can sit with a privacy-first specialist (Proton-style), a hyperscaler (AWS/GCP-backed), a bank (already trusted with money), a telco, or a vertical specialist (a health vault, a career vault). Because they hold

only ciphertext (above) and the vault is portable by re-upload, these are fungible: a bank and Proton are interchangeable hosts for the same data, and self-hosting (mode 1) is simply the user acting as their own provider. No provider is load-bearing; switching is a re-upload plus a pointer update, with exit always available. The market is what makes the hosted modes safe to choose - competition on price, privacy reputation, geography, and integration. Lexenne may run one such provider as a reference implementation; it does not own the market.

5.3 Agent

Fully fungible by design. An agent is any runtime that holds a capability token from the user. Examples:

- On-device small models (Apple Intelligence, on-device Gemini, local Llama variants)
- User's own hardware (local inference)
- Cloud frontier (Claude, GPT, Gemini API)
- Specialized agents (a health agent, a finance agent, a career agent - all running in parallel, all reading slices of the vault under different grants)

The user can run zero, one, or many agents simultaneously. Each agent's authority is bounded by its capability tokens; they can be revoked independently.

The agent never holds the user's signing key. All signed actions require either (a) the agent calling back to the user's device for an in-band signature, or (b) the agent operating under a pre-signed capability token with scoped authority.

---

6. Layer 1 in detail - the Personal Data Plane

Operations against Layer 1 (queries, mirrors, federations, ingestion) are SLF objects (§3) with substrate = facts.

6.1 L1a - Source Registry

A lightweight, mostly-local catalog of "what data exists about me, where, under what grant, last verified when."

Entries: - Source DID (the institution or person) - Data type hint (a broad, non-binding label - not a schema) - Grant reference (which capability token authorizes access) - Freshness policy (federate-on-demand, mirror-on-update, mirror-once) - Last successful access timestamp

This is the **map**, not the territory. It tells the agent where to look. The data itself is in L1b (mirrored) or at the source (federated).

On the type hint. The hint is a broad routing label - enough to tell an agent where to look - and nothing more. It does not fix meaning, restrict use, or impose a schema; interpretation happens at the lens, at retrieval (§1.3, §6.3), so one source can be read through many lenses later. The protocol does not own the hint lexicon. Like gates and predicates (§7.7), hints are federated: at most a tiny optional core, with implementations and domain bodies free to define their own labels, at whatever granularity and in whatever language a use case needs - one system's single "health" label may sit three levels deeper in another. How incoming data is matched to a hint is an implementation concern. This is the deliberate choice not to freeze semantics at ingestion (§1.3): the hint helps routing without locking in interpretation.

6.2 L1b - Semantic State

The durable, signed, provenance-rich facts about the user. **This is a semantic-state substrate, generalized.**

Each fact carries: - Content (the assertion) - Issuer DID (who claimed it) - Subject DID (the user) - `created_at`, `valid_at`, `invalid_at`, `last_seen_at` - `supersedes_id` (lineage chain for retractions and updates) - Source credential reference: the verifiable credential (VC) or signed envelope that delivered it - Confidence / trust tier - **Intrinsic gates** (signed metadata) - regulatory tags (for example HIPAA, GDPR, PII), audience restrictions, no-further-disclosure, expiry, issuer-imposed terms. These travel with the fact through every operation and are enforced by the query engine before any lens sees results. Standard vocabulary specified in §7.7.

Gates are intrinsic, not layered. Constraints on a fact (regulatory, audience, expiry, issuer terms) are properties of the fact itself, carried in signed metadata, never stripped at ingestion, mirroring, federation, or lens application. **Gate preservation is a substrate invariant.** HIPAA-tagged credentials cannot be exfiltrated by a lens that lacks HIPAA authority; GDPR-protected facts retain their right-to-be-forgotten obligation across providers; issuer-imposed restrictions ("not for use outside the EU") travel with the credential. The query

engine enforces gates before returning anything to a lens - lens code never sees what a gate excludes, and therefore cannot leak it. Detailed enforcement model: §7.

Gate authoring: who tags, who enforces, and what the protocol actually owns. The gates listed above are examples, not an authoritative or exhaustive list - like hints and predicates, the gate vocabulary is federated (§7.7), and no single party can own the long tail across jurisdictions. So the protocol's job is deliberately narrow: it guarantees gates travel with the fact, are never stripped, and are re-evaluated at every hop; it does not decide what a fact means or when a label applies. That decision sits with the parties best placed to make it.

- **Issued data carries issuer-signed gates.** A fact that arrives as a verifiable credential (a clinic's lab result) is signed by the issuer together with its gates - the HIPAA tag is part of what they sign - so the gate is as trustworthy as the issuer and bound cryptographically to the fact. This is the high-coverage, high-trust path, and the protocol's preference is for gates signed at source.
- **Self-entered and imported data is self-attested, tagged by the client at ingest.** When you key in your own details, or import a thousand contacts from a phone, there is no external issuer: you (your software, on your behalf) are the issuer, at the self-attested tier. A conforming client applies default gates here - default-deny for anything plausibly sensitive - rather than trusting a person to hand-label. Imported data about other people is the hardest version: the protocol can carry a third-party-PII gate, but whether you are a "controller" with obligations for those contacts is a legal question for the deployer, not something the protocol decides.
- **Whether a combination is PII is decided at the lens, not frozen at ingest.** A first name is not PII; a first name plus a home address often is; an email is PII in one jurisdiction and public in another. That determination is contextual and jurisdictional, so it cannot be a static per-field stamp. The substrate carries the raw gates; the lens, bound to a role and a jurisdiction (§9.3), evaluates at read time whether the specific combination being disclosed crosses a threshold - and it may consult other facts in the substrate to do so, including the subject's jurisdiction. This is the semantic-state principle (§1.3) applied to gates.
- **A fact can carry many gates, and modification means superseding, not editing.** PII and GDPR can both apply; gates compose, most-restrictive

wins. An issuer-signed fact is immutable - its gate cannot be stripped without breaking the signature. “Modifying” it locally writes a new, superseding, self-attested fact (Principle #4); the original and its gate are preserved, and a conforming lens honors the most-restrictive gate across the provenance chain. A user owns the data but cannot launder a regulatory gate off it by re-keying it.

- **Re-tagging on append is automatic, because evaluation is at read, not ingest.** A new field that turns a benign set into PII is caught at the next read, because the lens re-evaluates the live combination every time. Re-evaluation at every hop (a substrate invariant) is the recertification; there is no separate step to forget.

What remains genuinely open - and the position paper names this as the protocol’s most under-specified assumption - is **coverage**: gate authoring at scale has to be cheap, trustworthy, and high-coverage, and until that is measured on a real corpus, “every fact carries its gates” is partly aspirational. Default-tagging quality, the third-party-data responsibility question, and the contextual determination are the hard, unsolved parts (§11), named here rather than assumed.

Append-only. Agents and providers never overwrite. Retractions are new superseding facts. This is the substrate’s append-only discipline.

Bi-temporal. “When did this become true in the world” vs. “when did we learn about it” are separate axes. Already in the substrate model.

Storage policy is the user’s choice. Some facts mirrored locally (privacy, latency, durability). Some federated to source of record (Peloton stays system-of-record for bike data; vault holds the grant and metadata). High-value durable facts pulled into local mirror per user policy, even when federated - this is the liveness guarantee against source disappearance.

Ingestion from third-party copies. When a provider holds a Copy or Sync grant (§7.2) and pushes updates back, those updates land as new signed facts with provenance attributing them to the provider. The user’s substrate is the canonical destination for all derived facts. See §7.3 for the single-source-of-truth (SSOT) discipline.

6.3 L1c - Lens Registry

Per-context projections applied at retrieval. Not schemas. Not categories. **Lenses are queryable views over semantic state, shaped by the user's identity, current role, and current jurisdiction.**

A lens is: - A name (e.g., "Patient lens", "Employee lens", "Customer lens", "Health lens", "Career lens", "Chronicle/Timeline lens") - A query function over semantic state (facts matching predicates, with weighting) - Optional projection (how to present matching facts) - Optional aggregation (rollups, summaries) - Bound to a contextual role and jurisdiction at activation

Lenses are user-installable code. Like browser extensions. Anyone can publish a lens. Users choose which to install. Reputation and review mechanisms emerge in the lens marketplace; the protocol does not bless lenses.

One fact participates in many lenses simultaneously. A gym visit is in the health lens, the financial lens (membership cost), the chronicle/timeline lens (where I was on that date), the relationship lens (saw a friend there). Frozen ontology rejects this; lenses embrace it.

One lens at a time per active query. A user (or an agent acting under a grant) operates under one lens for a given query or run-time outcome. Switching lenses requires re-activation in the new context. This keeps the projection clear and the audit trail clean.

Lens governance is the lens marketplace's job, not the protocol's job. Lenses are extension code, not infrastructure. They can be wrong, narrow, opinionated, overlapping.

Lenses cannot bypass the gates the engine enforces - and carry a bounded responsibility for the rest. For any gate the engine can evaluate, the substrate filters first and only gate-passing facts reach the lens, so a lens never receives what a gate excludes and cannot disclose it. That part is structural, not cooperative, and it puts no regulatory burden on the lens author: they cannot leak a gate-excluded fact even if they tried, and they need not understand the regulation behind the gate. What the engine cannot pre-filter is contextual judgment - whether a particular combination of gate-passing facts is sensitive in a given jurisdiction, or whether a projection is minimal and within its purpose. There the lens author is responsible for projecting narrowly, but not solely and not to the letter of every regulation: the gate vocabulary (§7.7) carries the machine-checkable regulatory intent so the lens does not re-derive it, the deployer's assessment covers the system in context (the protocol is compliance-

enabling, not compliant), and over-disclosure is caught after the fact by receipts and the counterparty rating (§8.5). So it is genuinely "cannot" for the mechanical layer, and "should, and is held to it" for the judgment layer - not one blanket guarantee.

6.4 The unified Person pattern

A unified `Person` entity (`type: family | service_contact | medical | contractor`) - one row, four lenses. This is the lens architecture working at the domain level. The macro lift: `Self` at the substrate level, lensed as `Patient`, `Student`, `Customer`, `Employee`, `Borrower`, `Voter`, `Family Member` at retrieval.

This generalizes to the personal substrate ontology principle.

6.5 Resolution vs. deduplication - substrate-integrity discipline

The substrate ingests facts about people, things, and events from many sources. Two operations *appear* similar and are constantly conflated; conflating them silently corrupts the graph (Pauliusztin, §12):

Operation	Question it answers	Mechanism
Resolution	"What should we call this?"	Exact / fuzzy / semantic name matching against same-typed nodes; updates *canonical names only* - no graph merges
Deduplication	"Is this the *same real-world entity*?"	Embed full context; semantic + fuzzy similarity across the entire record

Surface-form name similarity is **not strong enough evidence** to merge entities. Apple (company) ≠ Apple (fruit). "Jensen Huang, CEO of NVIDIA" ≠ "Jensen Huang, doctor in Taipei." Resolution updates display strings; deduplication merges identity.

For people, our DIDs solve deduplication cleanly. Identity is anchored cryptographically, not by name matching. Two facts referencing the same DID describe the same person, full stop. Resolution (canonical-name handling for display, "Andrew Crenshaw" vs "Andy Crenshaw") is metadata work; *deduplication* is essentially free because the cryptographic anchor does it. This is a hidden win of person-granularity (§1.1) + DID permanence (§1.2).

For things, canonical identifiers carried by authoritative SLFs are required. A Honda service SLF carries the VIN (vehicle identification number). An ISBN carries a book. A DID-of-issuer + serial carries a credential. The substrate **requires** canonical identifiers in substrate types where such identifiers exist; without them, the substrate degrades to resolution-only and refuses to merge.

For facts, evidence-strength = permission-strength. The graduated outcomes from Pauliusztin map cleanly onto our trust tiers (§7.5, §11):

Trust tier of source	Match strength	Outcome
Trusted-List qualified issuer (eIDAS, AAMVA) + signed	≥ 0.95 equivalent	auto-incorporate; supersede prior with provenance chain
Rating-system mid-tier or peer-attested	0.85–0.95 equivalent	flag for user review via the app/lens; do not silently merge
Self-attested or unknown source	≤ 0.85 equivalent	new fact; never overrides existing facts

Weak evidence earns a new node. Strong evidence earns a merge. Uncertain evidence earns a queue surfaced through the app - never silent corruption.

What a merge is here, and why signatures survive it. “Merge” in an append-only, signed substrate is not the destructive node-combine of a mutable graph database. No signed fact is ever rewritten, so the worry “are the surviving record’s signatures still valid?” does not arise the way it would elsewhere. Two non-destructive operations happen instead, and both preserve every fact’s signature, issuer, provenance, and gates:

- **Entity association.** Recognizing that two records are the same real-world entity means pointing both sets of facts at one identifier (a DID for a person, a canonical id for a thing). The facts are not combined; they simply hang off one entity now. “Andrew” and “Andy” remain two separate signed name-facts, and a lens picks the canonical or most-trusted one for *display* at read time - that is resolution (top of this section), not a mutation. Relationships and timestamps are themselves signed facts or edges, each keeping its own provenance.
- **Supersession.** When a newer fact genuinely contradicts an older one, the newer signed fact stamps the older one’s `invalid_at` and links it by

provenance; the older fact stays in transaction time, unaltered and still validly signed, and drops out of valid-time reads. Bi-temporal supersession, not overwrite.

So a “surviving record” is not an amalgam carrying mixed signatures - it is a set of individually-signed facts associated with one entity, each still carrying its own signature and its own gates, which a lens composes at read (most-restrictive gate wins). Where there is no shared identifier to associate on, the substrate refuses to merge and surfaces the ambiguity rather than guessing.

A merge is a state change like any other, so consumers holding Copy or Sync grants learn of it through the same Write-back / Sync path and bounded-propagation guarantee as any other state change (§7.2, §8.6) - and because consumers key on the stable identifier rather than the display name, a resolution change usually needs no re-keying. The substrate needs no separate merge-publication mechanism; it rides the existing state-change rails.

Operational discipline (substrate implementation, not protocol). Ingestion pipelines should checkpoint each costly stage (LLM extraction, embedding, dedup-check) so retries don’t burn tokens recomputing work already paid for. This is a substrate-implementation property, not a protocol requirement.

---

7. Layer 2 in detail - the Capability & Grant Plane

The Grant Plane is where the substrate/lens/frame model becomes operational. Every grant is an SLF object (§3). Every agent action passes through this gate.

7.1 What a grant is

A grant is an SLF object whose payload authorizes specific operations. The minimum viable shape:

- **Audience** - who or what the grant is for: the receiving party (an agent, a third party, or the user themselves in a different context). The data subject / principal is the *issuer* (the user whose DID signs the grant, and the subject of the lens, §3); this `subject` = principal, `audience` = recipient split matches the companion IETF SLF substrate-grants draft and OAuth’s `sub` / `aud`.
- **Lens** - the contextual projection this grant operates within (the user’s identity, current role, current jurisdiction)

- **Allowed frames** - the set of run-time outcomes this grant authorizes
- **Grant type(s)** - what the grant permits the holder to do (§7.2)
- **Validity** - time bounds, conditions, revocation criteria, use-receipt requirements
- **Signature** - by the user's DID; revocable

Grants are not raw OAuth tokens. They are typed, scoped, time-bounded, and bound to a specific lens. Multiple grants can coexist for the same audience (an agent may hold grants for several lenses, each authorizing different frames).

The user-authorized, scoped, revocable, out-of-band sharing this describes is the model UMA 2.0 (User-Managed Access) already standardizes, and offline attenuation is the macaroons / Biscuit / UCAN lineage (§12). The grant plane composes over that base rather than reinventing it. What it adds is specific: authorization bound to the data through intrinsic gates rather than to a resource endpoint at a central authorization server, re-evaluated at every hop under monotonic narrowing (§7.4), plus the governance-typed action taxonomy and the HITL frame boundary (§7.2) that delegated-agent action needs. Holding a token is not the novel part (a UMA client can act for a party too); the action taxonomy and the frame boundary are.

What operates in this layer. The grant plane is a contract, not a particular piece of software: a grant format, the scope language (§7.8), the substrate → lens → frame → HITL evaluation order, and the receipt. Anything that speaks that contract participates - a library embedded in the user's vault, an API an application calls, a management UI the user runs to issue and review grants, or a sidecar deployed alongside a third-party autonomous agent so that agent can take part. Two roles stay fixed whatever the form. Issuance is the individual's alone: only the user's DID can sign a grant, so authority always originates with the person. Enforcement happens wherever the data is held at the moment of the operation - in the sovereign case the user's own vault, which can prevent over-disclosure structurally; behind a counterparty it is their engine, which the receipt holds accountable. So the grant plane can be anything that conforms, and a sidecar on someone else's agent is a first-class deployment. Conformance is what lets it participate; the enforcement locus - your vault versus a counterparty's engine - is what bounds the guarantee.

7.2 Grant types

The grant model recognizes that real-world data exchange is not always "user holds, third party queries." Many providers will continue to insist on storing their own copies because their existing systems are architected around it. The protocol supports this reality while preserving the user as the source of truth (§7.3).

The standard grant types:

Type	Provider behavior	Use case
Read	Provider queries on demand; does not retain	Live API access; bandwidth-sensitive operations
Read-and-cache	Provider may cache for performance; must refresh after a TTL (time to live)	High-frequency reads; performance-sensitive UX
Copy	Provider stores their own copy; user's substrate remains canonical	Legacy systems that require local storage; regulatory archival requirements
Write-back	Provider may push updates to user's substrate as signed facts	Lab results, transaction records, anything the provider generates about the user
Sync	Bidirectional updates between provider's copy and user's substrate	Active collaboration (medical records, financial accounts)
Action	Provider/agent may take specific actions on the user's behalf	Booking, purchasing, submitting, signing under user authority
Compound	A single grant combining several of the above	An active medical relationship: Copy + Write-back + Sync + (limited) Action

Each grant carries one or more types. The combinations chosen by the user encode the actual sharing arrangement: one-shot disclosure (Read, one frame, single use), ongoing relationship (Compound, many frames, time-bound), permanent reference (Copy with no expiry but revocable).

7.3 Third-party storage and SSOT discipline

When a grant includes **Copy** (or any storage-retaining type), the provider holds a working copy of substrate. **The user's personal substrate remains the canonical source of truth** (Principle #12). Three disciplines preserve this:

1. Working copies are not new substrates. A provider's Copy is a derived view, not an authoritative substrate. It carries provenance back to the user's substrate as its source. Reconciliation always flows toward the user's substrate as the canonical version.

2. Write-back is a grant term, made to cost the provider to skip. When a provider creates new facts about the user (lab results, transaction history, derived analytics), the grant obligates it to return them to the user's substrate as signed assertions, with provenance attributing them to the provider. The honest question is how that obligation is enforced, since the protocol cannot force a remote provider to behave (the same limit as gate enforcement and at-rest encryption). It is not compelled - it is made incentive-compatible and accountable. **Incentive:** write-back is tied to the receipt-as-precondition pattern - a provider that wants the user, or the user's agent, to accept and act on its output returns it with a receipt, so skipping write-back costs the provider the completed transaction. **Accountability:** a missed write-back is a grant-term violation that drops the provider's rating (§8.5) and is grounds for revocation. **Pull where observable:** where the user can independently see that an operation happened - they know a lab was ordered - they can pull the result rather than wait for a push. The honest residual is the same one that governs covert reads and receipt suppression: data a provider creates entirely on its own side and never surfaces cannot be compelled back by any message-passing protocol. So write-back is stronger than good-faith for everything the user can leverage, and accountability-only for what a provider keeps wholly to itself.

3. Reconciliation is append-only with provenance. If the user's substrate and a provider's copy diverge, both versions coexist as signed facts in the user's substrate, each attributed to its claimant. The user's substrate carries the complete history; lenses surface the most current or most trusted version depending on the use context.

Revocation ends use rights, and can carry an erasure request. The protocol cannot reach into a provider's storage to delete a copy directly, so it does two things instead. First, it ends ongoing use rights: after revocation the provider may no longer query the user, push updates back, or take actions, and

existing copies go stale. Second, where the user invokes the right to erasure (GDPR Art 17), revocation carries a signed deletion request the provider is legally obligated to honor - the protocol issues and receipts the request, the deletion is the provider's legal duty, and a failure to act is the provider's compliance violation, evidenced by the receipt and reflected in its rating. The lawful exception is a genuine retention obligation (financial or medical records, for instance), which overrides erasure under the same law. This is the compliance-enabling posture, not a claim that the protocol deletes data on someone else's servers: the protocol supplies the demand and the evidence, and the law supplies the obligation. (Erasure of data held in the user's own substrate is a separate matter, handled directly by the substrate's own erasure mechanism.)

This is a pragmatic concession: the protocol does not require providers to rearchitect their systems overnight. It requires only that whatever they hold respects the SSOT discipline. The protocol supports both the share-on-demand (Read, no Copy) pattern and the storage-retaining patterns (Copy/Sync), and meets providers where they are without prescribing which they adopt.

7.4 The substrate → lens → frame evaluation chain

Every action authorized by a grant is evaluated in this order. Each step can only narrow what the previous allowed:

1. **Substrate gates** (intrinsic, per-fact) - enforced first, non-negotiable. The query engine respects fact-level constraints (HIPAA, GDPR, audience restriction, expiry, no-further-disclosure) before any lens sees results. A grant cannot bypass them; they are properties of the facts, not properties of the grant.
2. **Lens projection** - the user's identity, current role, and current jurisdiction shape what facts the grant's lens can query at all. A grant for the "Patient lens" sees facts the patient role authorizes; the same facts viewed under the "Employee lens" may be invisible or differently filtered.
3. **Frame check** - does the specific run-time action fall within the grant's allowed frames? Inside the Patient lens, frames might include "schedule appointment", "review test results", "consent to records release". An action outside the granted frames is denied or escalated to HITL.

4. **HITL gate** - if the frame requires explicit user approval (per user preferences or grant terms), the agent's action pauses for in-band signature from the user's device.

The agent never reasons about regulation. It queries through its granted lens; substrate gates are enforced automatically by the substrate; the frame check is enforced by the grant plane.

7.5 Worked example - healthcare visit

To illustrate the grant types and SSOT discipline in practice. Healthcare is highly regulated and is not the v0 wedge for this reason; the example is illustrative only.

Setup. Andrew has a primary care network and is about to have an appointment. He has a personal substrate containing his health history, allergies, medications, and surgeries.

Pre-visit (one week before). The health network asks Andrew to confirm and share his current health summary. Andrew's agent issues a grant to the network:

- Grant type: **Copy + Write-back** (the network stores its copy and, honoring the grant's write-back term, returns the new facts it creates)
- Lens: Patient lens - the substrate is filtered to facts the patient role authorizes for healthcare disclosure
- Allowed frames: "pre-visit intake", "physician review", "lab order", "post-visit notes"
- Validity: through end of appointment + 30 days for follow-up
- Use receipts required: yes

The network ingests the disclosed facts into their electronic health record (EHR) system. Andrew's substrate notes the grant and what was disclosed.

During the visit. The doctor reviews Andrew's health history (within the network's copy, under the granted lens and frames). Bloodwork is ordered. Lab results come back indicating a potential new condition.

The network pushes the new lab results back to Andrew's substrate as signed facts (Write-back is part of the grant). Andrew's substrate ingests the results as new semantic-state entries with provenance signed by the network's lab, and with intrinsic HIPAA gates attached (the facts are now subject to HIPAA constraints whenever they leave Andrew's substrate).

Post-visit referral. Andrew's doctor recommends a specialist. Andrew (via his agent, within the Patient lens, in a "share with specialist" frame) issues a new grant directly to the specialist:

- Grant type: **Copy + Write-back**
- Substrate: filtered subset of Andrew's substrate - the lab results, the diagnostic hypothesis, and the relevant history
- Lens: Patient lens
- Allowed frames: "consultation", "additional testing", "specialist report"
- Validity: six months
- Use receipts required: yes

Note: the specialist did **not** receive the data from the primary network. Andrew is the SSOT; the grant goes from Andrew's substrate to the specialist.

Specialist visit. Specialist reviews. Orders additional tests. Generates a report. All new findings flow back to Andrew's substrate via Write-back.

Back to primary care (Andrew's choice). Andrew chooses to share the specialist's report back with his primary network. The existing primary grant's Sync provision (if configured) carries the new facts forward automatically; or Andrew issues a one-shot update grant. Either way, Andrew is the source - not the specialist sending to the primary directly.

What the SSOT discipline gave us:

- Andrew's substrate has the complete current picture: history, lab results, specialist report, primary care updates
- Each fact carries provenance: who claimed it, when, under what authority
- The primary network and the specialist each have working copies; neither is canonical - Andrew is
- If Andrew switches health networks, his substrate moves with him; new providers ingest from the user, not from old providers
- Conflicts (e.g., specialist disagrees with primary care interpretation) are visible in Andrew's substrate as competing signed facts; lenses can surface both

Healthcare is regulated tightly enough that this won't be the v0 wedge. But the structural pattern is the same for any domain where providers store copies of user data: SSOT lives with the user; provider copies are working copies; everything flows back to the user with provenance.

7.6 Why this is safer than the alternatives

- **Gates cannot be lost.** They live inside the data, signed by the issuer, traveling with every operation. There is no separate policy plane that can drift out of sync.
- **The lens marketplace is safer by design.** A third-party lens cannot bypass the gates the engine enforces - they are applied before the lens sees results, so a bad lens is contained automatically for that layer. The residual the engine cannot pre-filter (minimal, in-purpose projection of gate-passing facts) is the lens's responsibility, policed by receipts and ratings (§6.3, §8.5).
- **Agents are naive about regulation.** This is the only safe model - agents cannot accidentally violate constraints they do not know about.
- **Cross-jurisdiction composes cleanly.** A US-person-in-Italy-sharing-with-EU-provider has three frames overlapping; the intersection (most restrictive) applies. Frame composition is deterministic; no policy reconciliation engine needed.
- **Jurisdictional rigor scales by region.** German-issued credentials carry tighter substrate gates than Italian-issued ones; the protocol supports both without code changes. Each issuer/jurisdiction signs the constraints it wants enforced; the substrate carries them; lens engines respect them.
- **SSOT survives provider compromise.** When a provider is breached, the user's substrate is unaffected; only the provider's working copy is exposed. Revocation cuts off ongoing access; the user keeps everything.

7.7 Vocabulary via federated namespaces - the registrar-of-registries pattern

The protocol does NOT curate the long tail of gate types. A neutral steward role (§13), held by the maintainers today, curates a tiny core and the *meta-rules* for how others extend it. This is how IANA / MIME types / language tags / DNS root work, and it is the pattern that lets a small protocol survive being adopted across every domain on the planet (Principle #13 - Minimalism is survival). Each

authority hosts its own vocabulary at its own namespace by domain control - HIPAA gates at the health authority's domain, eIDAS gates at the EU's, a vendor's at the vendor's (see the discovery example below) - referenced at source, so no new central registry of gates is required and most of the model needs no central steward at all.

Three vocabulary tiers:

Tier	Curator	Cadence	Examples
Core	Neutral steward	Slow (annual review)	~10–20 universal gates: `personal-data`, `health-data`, `financial-data`, `minor-protection`, `audience-restricted`, `expires-at`, `purpose-restricted`, `one-shot`, `no-further-disclosure`, `disclosure-log`
Domain	Domain stewards (HHS-aligned for HIPAA; ENISA-aligned for eIDAS; AAMVA for mDL; ISO/sector bodies for finance)	Domain cadence	`hipaa:*` family, `eidas:*` family, `gdpr:*` family, `pci-dss:*` family
Vendor / individual	Anyone	Open registration by domain control	`vendor.example:custom-gate`, `did:web:lexenne.com:proprietary-gate`

Discovery doc declares supported vocabulary namespaces + version ranges, not individual gate names. A party intersects its supported namespaces with its counterparty's at exchange time. This is the pattern that lets new gates be added without every party in the world updating their discovery doc.

```
"supported_vocabularies": {
  "core-gates":   "https://vocab.slf.example/gates  v1.0..v1.3",
  "hipaa-gates":  "https://vocab.hhs.gov/hipaa-gates  v2.0..v2.1",
  "eidas-gates":  "https://vocab.enisa.europa.eu/gates  v1.0",
```

```
  "vendor.example": "https://example.com/our-gates  v1.5"
}
```

Same pattern applies to reason codes (in Receipt SLFs, §3.6) and **predicates** (in scope expressions, §7.8). Each is a typed value `{vocabulary: <URI>, term: <token>}` resolved against the party's declared namespace support.

The steward's role is structural, not curatorial. It defines: - How to claim a namespace (registration by domain control, no permission) - How to version a namespace (semver discipline, range negotiation) - How to deprecate a term (annotation, sunset period, fail-closed at sunset) - How to publish a vocabulary (canonical machine-readable schema, public retrieval)

The steward does **NOT** validate the semantic correctness of any namespace beyond the core. It is the registrar of registries.

Fail-closed on unknown gates and unknown vocabularies. Lens engines that encounter a gate from a vocabulary they don't support MUST refuse to render the affected fact, log the denial reason as `unsupported_vocabulary`, and surface the gap to the user/operator. This is what prevents new regulations from silently leaking data through old code. (§11.5 / §11.6 walk through the semver + capability negotiation + conformance suite consequences.)

7.8 Scope expression language for grants

A grant's `scope` and `constraints` fields are not free text. They are expressed in a small, versioned, non-Turing-complete language so that "does this scope allow this operation?" is statically decidable in bounded time.

Three constraints on the language:

1. **NOT Turing-complete.** No loops, no recursion, no arbitrary computation. Every scope expression must be statically analyzable.
2. **Tiny core operator set** (steward-curated, intentionally small): `AND`, `OR`, `NOT`, `EQUALS`, `WITHIN(timeRange)`, `MATCHES(predicate)`.
3. **Typed predicates from a versioned predicate vocabulary** (federated namespaces, same pattern as gates above). Examples: `predicate:health-domain`, `predicate:financial-aggregation`, `predicate:employer-verification`. Predicates are registered like gates; the steward role curates the smallest core; domain bodies and vendors extend.

Worked example:

```
scope:
  AND(
    MATCHES(predicate:employment-history),
    NOT(MATCHES(predicate:salary-disclosure)),
    WITHIN(start=2026-01-01, end=2027-01-01),
    EQUALS(audience.did, "did:web:linkedin.com")
  )
```

Reference parser/evaluator in the reference implementation serves as the oracle for ambiguous cases. The conformance suite (§11.6), once built, tests scope-expression evaluation against published test cases.

What actually needs a steward, and what does not. Most of the model needs no central steward: domain bodies and vendors host their own vocabularies at their own namespaces (domain control), and two parties resolve gates by intersecting the namespaces they each declare - the IANA / DNS pattern, which works peer-to-peer and degrades gracefully even if no central body exists. What does need an active steward is narrow but real, and best-effort maintainers cover only part of it. The low-load part - holding the tiny core (a handful of operators and ~10-20 core gates, blessed maybe once a year) and the meta-rules - maintainers can carry. The genuine dependency is arbitrating cross-namespace disputes and rejecting squatting: these need a recognized authority, not goodwill, and best-effort maintainers cannot credibly do them at scale. So the honest position is that the long tail is self-governing, the core is maintainer-holdable, and dispute-arbitration and anti-squatting are where a real steward (the §13 nonprofit, if and when it forms) earns its keep - named here, not assumed to be free.

7.9 What this section does not specify

- The exact wire format of grants - deferred to the protocol spec proper
- Cryptographic suite choices - deferred to implementation (point to HAIP 1.0)
- HITL UX patterns - a general-public design problem
- Inter-agent grant delegation - open question (§11.4)
- Conflict resolution UX when provider Sync copies diverge - app-level concern, resolved at the lens/app layer with the substrate's evidence-strength discipline (§6.5)

---

8. The bridge protocol (Layer 4)

The wire and trust layer between the sovereign substrate and the external world. **This is what we are actually building.** Every bridge interaction is an SLF (§3) over the wire formats below.

What it is, and where it lives. The bridge protocol is an open specification, not a service or a server: a thin profile over wire formats that already exist (SD-JWT VC, mdoc, OID4VCI/VP, HAIP; §8.2-8.3), plus a discovery document (§8.4), a trust model (§8.5), and the grant and receipt conventions. Applications, vault providers, institutions, and agents develop against it the way they develop against OID4VP today. It composes with a tool protocol like MCP (the Model Context Protocol) - an SLF capability can be exposed as a capability-gated MCP tool - rather than being an MCP server itself.

It is built once and shared, not rebuilt per organization. The spec is authored once and published openly under a permissive license (§13); each participant implements it once for itself; and any two conformant parties interoperate by reading each other's `.well-known/sovereign-agent` discovery document (§8.4), with no bilateral, per-counterparty integration. This is the SMTP property: the standard is written once, every provider implements it once, and any two can exchange without a custom connector between them. Avoiding N×M custom integrations is the point (Principle #13).

8.1 Two flows

Inbound: Provider → Vault "Here is a verifiable credential about your user."

Examples: Mt Sinai issues "Andrew completed lipid panel on 2026-04-12, value X, signed". Coursera issues "Andrew completed course Y on 2026-03-01, signed". An ex-employer issues "Andrew was employed Y/M/D to Y/M/D in role Z, signed". The Honda dealership issues "Andrew's vehicle had service event on date X, mileage Y, performed by person Z, signed".

These flow into the vault as semantic-state facts, with the credential as provenance, and intrinsic gates from the issuer preserved as signed metadata.

Outbound: Vault → Provider "Here is a signed assertion from your user (or your user's agent on their behalf)."

Examples: A loan provider receives “Andrew’s agent asserts: income > $X, signed by Andrew’s DID, backed by VCs A, B, C in vault”. A job board receives “Andrew’s agent asserts: holds credential D, signed”. A medical provider receives “Patient’s agent asserts: consent to share records A, B, C with provider E for purpose F, signed”.

8.2 Wire formats (EUDIW-aligned)

- **SD-JWT VC** (primary) - JSON-based, native selective disclosure, mandatory for EUDIW
- **ISO/IEC 18013-5 mdoc** (binary, high-assurance) - mandatory for EUDIW, used for government-grade credentials
- **W3C VC DM 2.0** (optional) - for non-qualified credentials, long-tail issuers, peer-to-peer attestations

8.3 Protocols

- **OID4VCI** - OpenID for Verifiable Credential Issuance (provider → vault)
- **OID4VP** - OpenID for Verifiable Presentations (vault → verifier)
- **HAIP 1.0** - High Assurance Interoperability Profile (security and privacy floor)

These exist. They are specified. They have interop tooling. We implement; we do not invent.

8.4 Discovery - `.well-known/sovereign-agent`

Both sides publish a canonical JSON document at `https://<host>/.well-known/sovereign-agent`. The discovery doc is one of the protocol’s small set of immutable schemas (Principle #13: every party in the world updates only when a major version bumps).

Where it lives and how it is trusted. `<host>` is each party’s own domain: a user’s vault, a provider, or a verifier each publishes its own discovery document on its own infrastructure, at a fixed, predictable path - the IETF `.well-known/` convention (RFC 8615). The idea is simple: to learn how to interact with a host, you fetch a small JSON file at a known location on that host, and it tells you where its endpoints are, what it supports, and which keys verify its signatures. (It is the same convention an OAuth or OpenID server uses for `/.well-known/`

`openid-configuration`, familiar to most web developers.) There is no central host and no registry; no steward hosts or maintains anyone's discovery doc. It is served over HTTPS, and its contents are bound to the publisher's DID and keys (the `jwks_uri` plus DID-anchored signatures, §8.5), so a reader verifies authenticity against the DID rather than trusting the transport or a central authority. Keeping it current is each party's own concern, and the schema is stable enough (Principle #13) that updates are rare.

```
{
  "protocol_version": "1.0",
  "did": "did:web:andrew.lex",
  "endpoints": {
    "inbound_credentials":   "https://andrew.lex/bridge/inbound",
    "outbound_presentations":"https://andrew.lex/bridge/present",
    "audit_log":             "https://andrew.lex/bridge/audit",
    "grant_status":          "https://andrew.lex/bridge/grants",
    "revocation_status":     "https://andrew.lex/bridge/revocation"
  },
  "supported_credentials": [
    { "type": "VerifiableCredential", "format": "sd-jwt-vc",
      "version": "draft-ietf-oauth-sd-jwt-vc-16" },
    { "type": "mDL", "format": "mdoc", "version": "iso-18013-5" }
  ],
  "supported_protocols": [
    "oid4vci/1.0", "oid4vp/1.0", "haip/1.0", "slf/1.0"
  ],
  "supported_vocabularies": {
    "core-gates":  "https://vocab.slf.example/gates  v1.0..v1.3",
    "core-codes":  "https://vocab.slf.example/codes  v1.0..v1.1",
    "predicates":  "https://vocab.slf.example/predicates  v1.0..v1.2"
  },
  "trust_anchors": {
    "trusted_lists": [
      "https://eu.eidas.europa.eu/trusted-list",
      "https://aamva.org/mdl/trusted"
    ],
    "rating_system_endpoint": "https://ratings.slf.example/v1"
  },
  "audience_jurisdiction": "US-CA",   // see §9.3 - self-declared
  "supported_receipt_modes": ["v1-counter-signed", "v2-zk-attestation"],
  "supported_anchor_protocols": [],   // see §8.10 - optional chain anchoring
  "public_keys": { "jwks_uri": "https://andrew.lex/.well-known/jwks.json" },
  "metadata": {
    "display_name":     "Andrew Crenshaw (personal)",
    "terms_of_service": "https://andrew.lex/tos",
    "privacy_policy":   "https://andrew.lex/privacy"
  }
}
```

A provider's discovery doc looks similar; a verifier's may omit `inbound_credentials`. The doc is the only place the protocol pins concrete field names; everything else is delegated to the vocabularies it references.

8.5 Trust model

- DID-anchored signatures (cryptographic root of trust)
- Trusted List for qualified issuers (government, regulated, enterprise tier)

- Reputation/web-of-trust for non-qualified issuers (peer, community tier)
- Selective disclosure mandatory (reveal only what's needed)
- Use receipts logged in vault (audit of what was disclosed to whom when)

Counterparty rating. For counterparties not on a Trusted List, reputation is the backstop. A public rating scores observed behavior: how promptly a party honors revocations, whether its receipts show over-disclosure, whether it suppresses receipts. A low score is a signal, not a gate - the protocol surfaces "integrate at your own risk" and leaves the choice to the user or their agent. This is the least mature part of the trust model: it is designed, not built, and its hard problems are open - resistance to Sybil and collusion (fake or coordinated counterparties gaming the score), who operates the rating and keeps it neutral (a steward function, §13), and the cold-start before there is enough signal to score anyone. It is the accountability backstop for what prevention cannot reach, named here rather than assumed.

8.6 Revocation

- Bitstring Status List v1.0 (W3C) or equivalent for credential revocation
- Per-grant revocation in the capability plane (Layer 2)
- Per-credential supersedes chain in semantic state (Layer 1b)
- **Propagation guarantee** - revocation effective by `revocation_timestamp + max(TTL, push_latency_ceiling)`. Push to known consumers is best-effort; consumers MUST also pull on each use if the grant's TTL has elapsed. This is the bounded-propagation contract.

8.7 Receipt modes - v1 counter-signed, v2 ZK-attestation

Receipts (§3.6) are required by the protocol. *Which mode* is negotiated between parties at exchange time, via the `supported_receipt_modes` field in their discovery docs.

Mode	Mechanism	Property	Tradeoff
v1 counter-signed	Both parties sign the receipt; both retain a copy	Maximum cryptographic non-repudiation; mutual evidence	Counter-party retains a permanent discoverable record of disclosure detail (legal-liability exposure for

Mode	Mechanism	Property	Tradeoff
			providers; subject to GDPR Article 5(1)(c) data-minimization scrutiny)
v2 ZK-attestation	Counter-party produces a zero-knowledge proof of gate-evaluation correctness without retaining the disclosure detail	Mutual proof of evaluation; minimal retention	Computationally heavier; depends on a viable ZK proof system for the gate-evaluation circuit

The protocol does not pick a winner. Both modes are first-class. A user can require v2 for sensitive disclosures; a provider can decline counterparties that don't support a mode they can legally honor. The conformance suite (§11.6), once built, tests both modes. This dual specification responds to two failure modes - a regulator ruling receipts non-compliant, and receipts becoming a liability - that both converge on the bilateral-retention model as the kill vector.

8.8 Grant invocation patterns

Two flows for how an outbound SLF carries the authorization that justifies it.

Standing grant (no HITL required at invocation time):

1. User previously signed a Grant SLF specifying scope (§7.8 expression language), audience, expiry, intrinsic-gate constraints.
2. Agent constructs outbound SLF; `frame.authorized_by_grant = <grant_SLF_id>`.
3. Recipient validates: (a) signature chain back to user DID; (b) grant SLF exists and is not on the revocation list (via `grant_status` endpoint); (c) outbound frame's scope ⊆ grant scope; (d) constraints satisfied.
4. Recipient emits Receipt SLF (granted or denied).

HITL-required (no standing grant covers this operation):

1. Agent detects no standing grant covers the requested operation.

2. Agent constructs outbound SLF with `frame.requires_user_approval: true`.
3. Vault prompts user; user signs an Approval SLF (a single-use grant), which the agent attaches inline in the outbound frame.
4. Recipient validates approval signature + scope; emits Receipt.

8.9 Standing-grant defenses, and where each is enforced

The most dangerous failure mode - a coordinated social-engineering attack against standing-grant UX (the "BenefitsBridge" scenario) - is met by defenses the protocol specifies and a conforming vault enforces. Be precise about where each one bites: some are structural in a conforming vault at the moment a grant is constructed (strongest in the sovereign case, where the user runs the vault); the policy values behind others are jurisdictional, not protocol constants; and behind a third-party-hosted vault, conformance plus the rating (§8.5) is what holds, since the protocol cannot reach into someone else's software.

1. **Audience identity is checked at construction, not by display name.** A grant whose `audience.display_name` does not match the audience DID document's metadata is invalid, and a conforming vault refuses to construct it - foreclosing fake-portal attacks that show a plausible name unrelated to the actual recipient DID. This is a construction-time check: structural in a conforming vault, and in the sovereign case (the user's own vault) it cannot be bypassed.
2. **Unbounded scope is rejected structurally; which bundles are acceptable is not the protocol's call.** A scope that resolves to "everything", or that names no specific field families, is invalid - a mechanical check any conforming vault makes. But the protocol cannot judge that `bundle:cdc-immunization-record` is acceptable while `bundle:lifetime-heart-record` is not; a lifetime heart record may be entirely legitimate. What counts as an acceptable named bundle is a domain-and-jurisdiction judgment carried in the federated vocabulary (§7.7), not a protocol whitelist. The protocol forbids unbounded grants; the catalog of bounded bundles is governed where the domain expertise lives.
3. **Lifetime is capped, but the cap is a jurisdictional value, not a protocol constant.** A standing grant carries a maximum lifetime, and renewal requires a fresh user-signed approval. The number is not fixed by the protocol: one jurisdiction may cap it at thirty days, another at seven, and

the applicable cap travels as a gate (§7.7, §9.3) that the conforming vault enforces. Hardcoding a single number would be the frozen-semantics mistake again - the protocol provides the cap mechanism, jurisdictions set the value.

4. **Concentration warnings are a conformance behavior - checkable and accountable, not something the protocol can make a host do.** A conforming vault surfaces a warning when one audience accumulates an outsized share of a user's standing grants (an early signal of harvesting at population scale). The protocol cannot force a third-party vault to implement this; what it can do is make it a conformance requirement, let an attested runtime prove a vault runs the conforming build, and let the rating (§8.5) and receipts catch a vault that does not. On the user's own vault it is simply enforced.

So these are conformance requirements with structural force in a conforming - especially sovereign - vault, parameterized by jurisdiction where policy varies, and backed by attestation and rating behind a counterparty. They are not magic the protocol imposes on every host. The conformance suite (§11.6), once built, is specified to test for them.

8.10 Optional anchoring to a public blockchain

The bridge protocol is **blockchain-neutral by default** - the "chain" here is a public blockchain (a distributed ledger). SLFs are signed and verified by DID resolution; no blockchain is required for any normal operation. This is foundational: hard coupling to a blockchain (the Self Protocol / Celo model) is the deepest philosophical divergence between adjacent personhood protocols and us (§12). The position is not anti-blockchain; it is that the protocol must not depend on one.

That said, *optional* chain anchoring is a supported protocol that adopters can leverage where it adds value. Two patterns:

Pattern	When appropriate	When NOT
Provenance anchor (additive)	An SLF requires extra durability of provenance (e.g., a multi-decade asset history; a one-time, legally significant attestation). Content-hash + signature published to a public	Routine operation. The cost - transaction fees, an external dependency, and the reputational and regulatory baggage a public blockchain carries - is

Pattern	When appropriate	When NOT
	blockchain serves as an immutable timestamp witness.	wrong for the common case.
Cross-substrate timestamp service	Two parties need a neutral, censorship-resistant timestamping primitive for high-stakes audit (regulatory disclosure, contractual milestone).	Internal vault-to-vault federation, where a simpler timestamp service suffices.

A party advertises chain-anchor support in its discovery doc via `supported_anchor_protocols`. Counterparties may require, accept, or reject anchored SLFs based on their own policy. **An SLF that lacks an anchor is fully valid** under the bridge protocol; an SLF that has one carries extra provenance.

Why this is never required. Anchoring introduces a dependency on the blockchain's continued availability, governance, and economics. The Self Protocol's experience (§12) - being structurally coupled to Celo - illustrates the cost of making a blockchain foundational rather than optional. We use blockchain anchoring selectively, without becoming blockchain-dependent.

---

9. EU Digital Identity Wallet alignment

The EU Digital Identity Wallet (EUDIW) is a strong, well-timed alignment for this architecture: a mandated identity-and-credential layer, arriving on a fixed timeline, that the protocol can compose with directly rather than reinvent. It is an opportunity to build on, not a dependency - the protocol does not require it, and it is one path to scale among others, not the single thing that makes the rest achievable. The sections below describe how SLF aligns with it where it lands.

9.1 The mandate

- Regulation (EU) 2024/1183 in force
- All 27 EU member states must offer eID Wallet to every citizen, resident, business by **end of 2026**
- Cross-border interoperability legally required, not optional

- ARF v2.8, 31 implementing acts, HAIP 1.0 published

9.2 What's still ours to specify

- The SLF primitive (§3) - EUDIW does not specify a uniform operational shape
- The agent capability and grant model (EUDIW does not specify agent delegation)
- The semantic state + lens model (EUDIW is a credential drawer, not a knowledge substrate)
- The federated source registry (EUDIW credentials are held; ours are held *or referenced*)
- The bridge discovery protocol (`.well-known/sovereign-agent`)
- The lens marketplace ecosystem
- The intrinsic gate vocabulary and substrate→lens→frame evaluation chain (§7)
- The grant types and SSOT discipline for third-party storage (§7.2–7.3)

This is exactly the right slot. The EU has standardized identity and credential exchange; we specify agent authority, persistent reasoning, sovereign-storage discipline, and the operational primitive that ties it all together.

9.3 Jurisdiction-per-operation - two-field model

Jurisdiction is a property of *each operation*, not of the user. Users move; regulations attach to the operation context, not to a daily-maintained user-identity field. The protocol carries two jurisdiction values in every Frame:

Field	Source	Derivation
`subject_jurisdiction`	User's vault/app	Priority: (1) explicit user preference; (2) app-level context (geolocation hint, vault provider's home jurisdiction); (3) last-known-good cached value; (4) `unknown` → fail closed on jurisdiction-sensitive gates

Field	Source	Derivation
`audience_jurisdiction`	Audience's discovery doc	Self-declared by the counterparty in `.well-known/sovereign-agent`. We do not verify the audience's actual physical location. If they lie, that is their compliance violation; the counterparty rating (§8.5) is the intended backstop for repeat offenders - designed, not yet built.

Gates address whichever jurisdiction(s) they need:

Gate class	Resolves against
Patient-rights, data-subject-rights, individual-rights	`subject_jurisdiction`
Controller-obligations, covered-entity rules, custodian duties	`audience_jurisdiction`
Cross-border transfer (e.g., GDPR Article 44, US export controls)	BOTH (composite rule)

Worked example. A US patient using a German app to share lab results with a Berlin hospital: - `subject_jurisdiction = US`, `audience_jurisdiction = DE` - HIPAA-patient-rights gate evaluates against US → patient retains US rights - GDPR-controller-obligation gate evaluates against DE → hospital has DE obligations - Cross-border-transfer gate evaluates BOTH → patient transferring into EU is allowed; hospital receiving from outside EU has fewer additional obligations than vice versa

The protocol owns: carrying the two fields, defining derivation priority, defining how gates declare which jurisdiction they address. **The protocol does NOT own:** detecting jurisdiction (app-layer), maintaining the user's location (vault-layer), interpreting specific national law (lens-layer, via vocabulary).

Failure modes (intentional design): both unknown → fail closed; conflict with gate expectations → gate denies; user in transit → `subject_jurisdiction` may be `in-transit`; lens engines treat as ambiguous and may degrade safely with audit note.

What we don't yet know. The above is the design *posture*; the hard cases (user actively in transit during a high-stakes operation; audience lies about jurisdiction and is only caught post-hoc via the rating system; subject and audience jurisdictions impose mutually incompatible obligations on a single fact) are genuinely open. The protocol survives them by failing closed, but failing closed is not the same as producing the *right* answer. Real-world pressure-testing (§11.12) is the only way to learn where this design breaks. Acknowledged as one of the protocol's most difficult unsolved problems.

---

10. What's out of scope (v0.1)

These are important; deferring keeps the architecture shippable.

- **Recovery deployment and consumer UX.** The recovery protocol core is designed and prototyped (§5.1); what is deferred is binding shares to platform hardware, real cloud-share custody, the total-loss UX, and testing with non-crypto users before general-public release.
- **Platform (Apple/Google) adoption dynamics.** A strategic question; not load-bearing for the protocol design itself.
- **Vault-provider business model.** Market dynamics, pricing tiers, regulatory positioning. Important at general-public scale, not for this architecture.
- **Cross-jurisdiction legal framework.** GDPR, HIPAA, CCPA, eIDAS, equivalents - the protocol's job is to enable compliance, not to specify it. (See §9.3 for how jurisdiction is carried; §12 for the mandates the protocol composes with.)
- **Implementation details.** Specific cryptographic suites, performance characteristics, storage formats below the wire - separate spec documents.
- **Agent reasoning and model internals.** How the agent plans, reasons, and which frontier model it runs are out of scope and deliberately model-agnostic (Principle #5). The protocol governs the agent runtime's (Layer 3) authority and accountability, not its cognition.
- **The lens marketplace governance and reputation system.** This architecture specifies that the marketplace exists; not how it's governed.

- **Global regulatory landscape detail.** Comparison of EU, US (state mDLs), UK, Australia, Canada, Singapore, India, Switzerland and the full regulatory-mandate alignment scan; not duplicated here.
- **SLF + Bridge Protocol conformance test suite.** Specified in principle (§11.6); building the actual suite is implementation work, not architecture.
- **Collusion between issuer and user to issue false credentials.** Out of scope by Principle #17. The protocol does not police institutional honesty; it surfaces trust tier and provenance for downstream evaluation.

---

11. Open questions

1. **Recovery for the general public.** *Model decided and prototyped (§5.1):* FROST threshold signing (2-of-3 default, 3-of-5 high-assurance), cooldown-then-probation, receipt-audited, key never reconstructed - all mechanically tested in a reference implementation. *Still open:* platform-hardware binding (Secure Enclave / StrongBox) and real cloud-share custody; the consumer UX for the total-loss case; and the regulatory posture under nation-state compulsion. A prototyped core with deployment and edge-case UX remaining, not a blank space.
2. **Spoofing prevention / protocol integrity.** Concrete defenses landed in v0.1: audit receipts as first-class output (§3.6); v1/v2 receipt modes (§8.7); standing-grant defenses (§8.9); conformance suite (§11.6); cryptographic protocol review. Lens sandbox specifics (WebAssembly isolation, declarative-first) remain implementation detail. *Still open:* any spoofing class not addressed by the above defenses; surface during conformance-suite construction and adversarial review.
3. **Vault provider verification.** *Approach:* the steward does not bless individual providers; verification rests on the counterparty-rating mechanism (§8.5) - audit and transparency-log inputs, “integrate at your own risk” below a threshold. *Open:* that rating mechanism is itself designed, not built (§11.7).
4. **Inter-agent communication and HITL friction.** *Decided:* always through the user’s grant plane (no direct agent-to-agent), to preserve the audit spine. *Open:* the friction curve for low-stakes agent-to-agent operations (e.g., user’s agent making a vehicle repair appointment via a

dealership receiving agent) - at what point does HITL friction become unacceptable, and how do we let users expand the autonomous envelope safely (Principle #9)? Pressure-test scenarios needed during early adoption.

5. **Intrinsic gate vocabulary governance.** *Approach decided:* federated namespaces (steward-curated tiny core + domain extensions + open vendor extensions, §7.7); semver + capability negotiation; fail-closed on unknown. *Still open:* the operational details of how a domain steward is admitted, how vocabulary disputes are arbitrated, and what the publication-channel SLA (service-level agreement) looks like.
6. **Bridge Protocol conformance test suite.** *Approach decided:* published suite + reference implementation as oracle (walkthrough captured during v0.1 design); 5 seed test cases drafted. *Still open:* actually building the suite and the governance for adding test cases over time.
7. **Grant revocation propagation (resolved) + counterparty rating (designed, not built).** *Revocation propagation - resolved:* best-effort push plus mandatory pull-on-use after TTL, with a bounded guarantee `revocation_timestamp + max(TTL, push_latency_ceiling)` (§8.6). *Counterparty rating - open:* the mechanism is described in §8.5 (score from receipts and revocation behavior, "integrate at your own risk"), but it is designed rather than built, and its hard parts - Sybil and collusion resistance, neutral operation as a steward function (§13), and cold-start - are unsolved.
8. **Decision-trace standalone spec.** Could a SLF decision-trace artifact (a sub-spec of audit receipts) ship as a standalone EU AI Act Article 50/86 compliance artifact *before* the full SLF protocol lands?
9. **DIATF / EUDI trust-list certification pathway.** What's the concrete path to becoming a recognized trust-list participant under UK DIATF and EU eIDAS / EUDI? Does the protocol's steward become an Approved Issuer, or does it sit at a different layer? Affects positioning.
10. **Visa TAP / ERC-8004 / Stripe MPP / OpenAI ACP / Google UCP positioning.** Several agent-commerce protocols are converging on a narrow slice of agent attribution (commerce + payments). SLF is broader (any agent action under grant). How do we frame the difference without inviting "you do everything" responses, and where does composition with these protocols make sense vs. competition?

11. **FIDA-style dashboard shape.** EU FIDA mandates a consent-management dashboard for financial data. What does the user-facing SLF dashboard look like at MVP, and how does it compose with EU-mandated dashboards rather than duplicating them?
12. **Jurisdiction failure-mode pressure-testing.** The two-field jurisdiction model (§9.3) is the design posture; the *hard cases* are open. Three classes need real-world pressure-testing before general-public release, ideally during early adoption with a skilled early-adopter community: (a) user actively in transit during a high-stakes operation (subject_jurisdiction is genuinely ambiguous); (b) audience misrepresents its jurisdiction and is only caught post-hoc - what's the remediation when a multi-month standing grant has been operating under bad assumptions; (c) subject and audience jurisdictions impose mutually incompatible obligations on a single fact (most-restrictive intersection still doesn't yield a clean answer). Fail-closed is the protocol's bailout but it is not the right answer; this is one of the protocol's most difficult unsolved problems.
13. **Gate authoring at scale.** The most under-specified assumption in the protocol, and the one the position paper flags as well: "every fact carries its gates" needs a tagging step that is cheap, trustworthy, and high-coverage (§6.2). *Open:* default-tagging quality for self-entered and imported data; the contextual, jurisdictional determination of when a combination is sensitive (a combination, not a field); responsibility for third-party data a user imports; and a measured coverage figure on a real corpus. Issuer-signed-at-source is the high-trust path; the long tail is unsolved.

---

12. References

Foundational thinking

- Gopinath, A., et al. - "Are We Learning the Wrong Bitter Lesson?" (Sentra, 2026), and the group's formal results in "The Price of Meaning" (arXiv:2603.27116) and "The Geometry of Forgetting" (arXiv:2604.06222). Semantic state vs. ontology lens framing, with the formal argument for why frozen-embedding memory degrades. Adopted in Principle #3.
- Sutton, R. - "The Bitter Lesson" (2019). What general methods + computation actually beat (and what they don't).

- Berners-Lee, T. - Solid Project. Sovereign substrate without an agent layer (cautionary lesson).
- Pauliusztin - Resolution vs. Deduplication discipline (X/Twitter, 2026). Separation of naming from identity; evidence-strength = permission-strength. Adopted in §6.5.
- Bernhardsson, E. - “Software companies buying software: a story of ecosystems and vendors” (2026-02-25). Deepening vendor stack; factor-out thesis; open spec + commercial services pattern. Informs §13 business posture.
- Capability and credential lineage - AWS Cedar (Cutler, J. W., et al., *A New Language for Expressive, Fast, Safe, and Analyzable Authorization*, OOPSLA 2024; arXiv:2403.04651) for an analyzable, decidable scope language (§7.8); macaroons (Birgisson et al., NDSS 2014), Biscuit, and UCAN for signed, scoped, time-bounded, attenuable-offline capability grants (§7). SLF composes over these rather than inventing a policy language.
- Certificate Transparency (IETF RFC 9162, 2021) - the hash-chained transparency-log tradition behind the audit-receipt chain (§3.6, §8.5).
- Remote attestation (IETF RFC 9334, RATS, 2023) - the attestation model the design names as the path to provable honest evaluation behind a counterparty (§5.2, §8.9).
- Threat-modeling vocabulary - STRIDE (security) and LINDDUN (privacy); OWASP Top 10 for LLM Applications (2025) for agent-specific risks. Applied in the companion THREAT_MODEL.md and §7.6.

Standards we adopt

- **Identity and data model:** W3C DID Core 1.0; W3C Verifiable Credentials Data Model 2.0.
- **Credential formats:** IETF SD-JWT VC (draft-ietf-oauth-sd-jwt-vc; builds on SD-JWT, RFC 9901); ISO/IEC 18013-5 (mdoc / mobile driving licence) and ISO/IEC TS 18013-7 (online presentment).
- **Exchange protocols:** OpenID Foundation OID4VCI 1.0, OID4VP 1.0, HAIP 1.0.
- **Revocation:** W3C Bitstring Status List v1.0.

- **Cryptography and encoding:** EdDSA / Ed25519 (RFC 8032); JWS (RFC 7515) and JWK (RFC 7517); JSON Canonicalization Scheme (RFC 8785); COSE (RFC 9052) / CBOR (RFC 8949), under the mdoc format.
- **Discovery:** `.well-known/` URIs (RFC 8615).

Adjacent protocols and prior art (compose with, do not replicate)

- **Self Protocol (Celo)** - proof-of-personhood + selective KYC attribute disclosure using ZK proofs over passport/national-ID. Founded by Reinsberg, Olszewski, Nakagawa (ex-Celo); $9M seed (Greenfield, Nov 2025); Google Cloud partnership (July 2025; GA Jan 28 2026). Adopted as a personhood and government-attribute issuer. Architectural philosophy diverges (chain-anchored vs federated substrate) - we compose, do not replicate.
- **Self Connect / `FederatedAttestations` contract (Celo)** - multi-issuer attestation registry with ODIS (Oblivious DID Service) OPRF (oblivious pseudo-random function) blinding for identifier obfuscation. Prior art for portions of our grant plane; OPRF blinding pattern worth evaluating for adoption.
- **ERC-8004 / Celo Agent Visa** - first Ethereum-standard agent identity (deployed Jan 29 2026) + tiered trust system (March 2026). Adjacent to SLF agent runtime + grant plane. Deferred deep dive.
- **India Account Aggregator + DPDP Consent Manager** - production analog for capability-grant + consent-receipt patterns at population scale (2.6B+ accounts). Distribution channel via DPDP CM positioning, not competition.
- **Visa Trusted Agent Protocol (TAP), Stripe MPP, OpenAI Agent Commerce Protocol, Google AP2 (Agent Payments Protocol; donated to FIDO, 2026), x402** - narrow agent-commerce and agent-payments primitives. Positioning study open (§11.10).
- **MCP, A2A** - integration substrates for agent communication; SLF composes with them, does not compete.
- **KERI / ACDC / vLEI (Trust Over IP, GLEIF)** - ledgerless key-event identity with native credential chaining (ACDC) and cooperative two-party delegation; vLEI is the organization-scale deployment. Prior art for the

grant-chain and receipt spine, and an alternative substrate-identity path to the SD-JWT VC wire we adopt. Compose, do not replicate.

- **UMA 2.0 (User-Managed Access, Kantara)** - the OAuth extension for user-authorized, scoped, revocable, out-of-band third-party access. The closest shipping, person-centric analog to the grant plane (§7); SLF adds gates bound to the fact, substrate-lens-frame narrowing, and payload-free receipts. Compose and position against.
- **OAuth Token Exchange (RFC 8693) + Rich Authorization Requests (RFC 9396)** - the mainstream delegated-authorization mechanism (`act` / `may_act` delegation chains; fine-grained `authorization_details`). A grant can bind to these at the OAuth edge; the grant itself is the capability-token lineage, not raw OAuth (§7.1).
- **ISO/IEC TS 27560 consent records + Kantara consent receipts (W3C DPV)** - the standardized, machine-readable consent-receipt structure behind GDPR / DGA / DPDP consent managers. The receipt (§3.6) should profile this rather than invent a parallel format (Principle #10).
- **Data spaces - IDSA Dataspace Protocol, Gaia-X, Eclipse Dataspace Components** - the EU framework for sovereign data exchange with ODRL usage-control policies that travel with the data; the institutional analog of intrinsic gates. Compose at the org-to-org boundary.
- **FIDO Agentic Authentication (AATWG)** - delegated-credential and intent-bound-assertion work atop WebAuthn for non-payment agent actions. The nearest standards-body effort on agent authority; track and align.
- **Also tracking** - Grantex (an IETF-draft grant/receipt/revocation model close to this one); AIP (Agent Identity Protocol, arXiv 2026); SPIFFE / SPIRE (workload identity under the agent runtime); the personal-data-store cohort beyond Solid (MyData, Verida, Inrupt); and the agent-memory layer (Letta, mem0, Zep), convenience memory without sovereignty or audit.

Regulatory context

- Regulation (EU) 2024/1183 - European Digital Identity Framework.
- ARF v2.8 - EUDI Wallet Architecture Reference Framework.
- Commission Implementing Regulation (EU) 2026/798 - wallet enrolment.

- AAMVA mDL Implementation Guidelines (US state DMV convergence on ISO 18013-5).
- CA DMV mDL program (CA.gov, 2026) - California reference implementation.
- UK Digital Identity and Attributes Trust Framework (DIATF), GOV.UK Wallet 2026 rollout.
- Switzerland E-ID law (2024) - non-EU jurisdiction adopting EUDIW-compatible stack.
- EU AI Act - Articles 50 + 86 (transparency, decision attribution; August 2026 effective). Relevant to the SLF agent-action-under-grant primitive.
- CFPB §1033 (US, April 2026 staggered) - financial data third-party authorization.
- CMS-0057-F (US) - health-data interoperability and consent.
- India DPDP Act 2023 + DPDP Rules 2025 - Consent Manager architecture as a distribution channel.
- EU FIDA (Financial Data Access regulation) - consent management dashboard mandate.
- EU Digital Product Passport (ESPR; Battery DPP February 2027) - substrate-as-DPP-registry overlap with consumer asset-provenance use cases.
- CCPA ADMT (California Automated Decision-Making Technology regulations) - agent-decision attribution.
- Maryland MODPA - minor-protection gate vocabulary alignment.
- AMLR (EU Anti-Money Laundering Regulation) - perpetual KYC primitive.

Governance precedents

- Mozilla Foundation / Mozilla Corporation - nonprofit-owns-for-profit structure, studied for the ordering (nonprofit primary). A cautionary tale on outcomes, though: roughly fifteen years of financial dependence on a single counterparty (Google) and steady share decline make it an example of capture, not of mission preservation under commercial pressure (§13).

- Judy Faulkner / Epic Systems - Purpose Trust split-vote / split-value governance. Studied as precedent for mission-locked stewardship resistant to acquisition pressure.
- Apache Software Foundation - open-spec stewardship by neutral 501(c)(3); reference for governance-charter discipline.
- Linux Foundation (incl. LF Decentralized Trust and the OpenWallet Foundation) - industry-consortium neutral home for protocols; the realistic neutral-host path for SLF.
- OpenAI nonprofit + for-profit cap - what to *avoid* (cautionary: for-profit dominant, mission subordinate is the wrong order).

---

13. Stewardship

The protocol exists to give autonomy and sovereignty to *individuals*. That goal cannot survive the protocol becoming the property of any single commercial entity. The governance posture follows from that - without committing to standing up an institution before one is warranted.

13.1 Open, and adoptable without its author

The spec, the vocabulary registries (§7.7), the conformance suite (§11.6), and the reference implementation are published openly under permissive licenses. The standards-track work (the grant draft) goes to the relevant existing bodies (the IETF); the vocabulary registries follow the IANA registrar-of-registries pattern; and the maintainers run the small core today. This is deliberately the lightweight path: it makes the protocol adoptable without its author, now, without a new institution in the critical path.

Lexenne builds commercial products on top of the protocol, and may offer hosted vault and agent-runtime services, lens-marketplace facilitation, and integration work. It operates on the protocol under the same conformance discipline as any other implementer, with no privileged access. The business is the services and certifications around an open spec (Bernhardsson, §12); the commercial path does not depend on closing the spec.

13.2 The neutral-steward role

Some functions are best held by a neutral steward rather than any single implementer: the federated-namespace meta-rules, admission of domain stewards, arbitration of namespace disputes, the rating system (§8.5), the public trust-list endpoints, and the SLF trademark. Today the maintainers hold these on the IANA model. The end-state - a dedicated, mission-locked nonprofit that holds them so the mission survives commercial pressure on any single implementer - keeps the nonprofit-primary ordering, not OpenAI's inversion (§12). Mozilla is the reference for that ordering, though its long single-counterparty financial dependence is a caution, not a success to emulate. The realistic neutral-host forms are an existing foundation - LF Decentralized Trust, the OpenWallet Foundation, or a W3C Community Group / IETF track - rather than a standalone sister entity, with the Inrupt experience as the base rate to plan for. It is reserved for when scale or a funding source warrants it, not a precondition and not a present commitment. Naming the role does not require standing up an institution; the protocol works with the maintainers in that seat until one is justified.

Whoever holds the role, it owns the *meta-rules*, not the content of every extension (Principle #13). It does not validate the semantic correctness of any namespace beyond the tiny core, bless individual vault providers, agent runtimes, lens engines, or apps, take a position on which use cases are legitimate (Principle #17), operate any commercial service, or issue or hold user credentials, grants, or data.

13.3 Licensing

- **Spec:** Apache-2.0 (with the option to relicense the spec text CC BY 4.0 if pursuing a W3C/IETF track).
- **Reference implementation:** Apache-2.0.
- **Conformance suite source:** Apache-2.0.
- **Trademarks:** the "SLF" mark is held by whoever holds the steward role (the maintainers today; §13.2); commercial use requires a trademark license, free for conformant implementations and revocable for misuse.

The spec is the free public good.

This is a working architectural design, not a final specification. The protocol core (SLF) is implemented in slf-core; this broader agent architecture will move toward a protocol specification as the reference deployment exercises the design.