The Capsule: A Sovereign Encrypted Process for Web5 Computing

Zach Kelling

Lux Industries

2025

Abstract

A Capsule is the fundamental unit of Web5 computing: a sovereign encrypted process that binds an identity to a local data vault, a capability system, a sync topology, and a compute policy. Every user, organization, AI agent, workspace, and device gets one. The vault is an encrypted SQLite database (AES-256-CBC via SQLCipher), synchronized offline-first through CRDTs, anchored to Lux for trust—identity roots, key policy, threshold controls, and audit receipts. This paper gives the formal definition of a Capsule, proves its security properties under standard cryptographic assumptions, classifies the three application classes it enables (local-first private, threshold-controlled, and confidential compute), and specifies the trust kernel that Lux provides. The thesis: Base owns state, Lux owns trust, providers own service quality, users own keys and exit rights. The Capsule is not a decentralized database. It is the personal computer of Web5.

1 Introduction

The dominant crypto thesis of 2017–2023 treated the blockchain as a universal substrate: database, server, and compute engine collapsed into one replicated state machine. Every document, every message, every user action was supposed to live on-chain. The result was systems that were simultaneously too slow for applications, too expensive for storage, and too public for privacy.

The failure was architectural. Blockchains are excellent trust planes—they produce globally verifiable proofs about who controls what keys, what policies govern those keys, and which state transitions have been authorized. They are terrible data planes. Storing a user's notes, messages, or collaborative documents on a replicated ledger is the wrong abstraction at every level: cost per byte, latency per read, privacy per query.

The local-first software movement [1] identified the correct alternative: applications that work offline, store data locally, and synchronize through conflict-free replicated data types (CRDTs). But local-first software without a trust layer has no answer for identity, key recovery, capability delegation, or auditability. It produces isolated islands of encrypted state with no shared coordination.

Web5 is the synthesis. The data plane is local: encrypted SQLite vaults per principal, synchronized as ciphertext via CRDTs, decrypted only on the owner's device. The trust plane is Lux: identity roots on I-Chain, key policies on K-Chain, threshold reveals on T-Chain, MPC signing on M-Chain, and audit anchors on the primary network. Providers—relay operators, storage hosts, recovery agents, compute markets—are optional, replaceable, and never data owners.

The Capsule is the unit that makes this concrete. This paper defines it formally, proves its properties, and specifies the three classes of applications it enables.

Contributions.

(i) Formal definition of the Capsule as a seven-tuple (Section 2).
(ii) Classification of three application classes with distinct trust requirements (Section 3).
(iii) Specification of the Lux trust kernel across four purpose-built chains (Section 4).
(iv) Security analysis under the post-quantum threat model with tiered cryptographic guarantees (Section 6).
(v) Design of agent capsules—AI agents as sovereign processes with auditable memory and bounded capabilities (Section 8).
(vi) Honest accounting of unsolved problems (Section 10).

2 The Capsule Model

Definition 2.1 (Capsule)**.** *A **Capsule** is a seven-tuple*

$$\mathcal{C} = (V, \mathcal{I}, \Gamma, \Sigma, \Pi, \mathcal{A}, \mathcal{P})$$

where:

- V *is the **vault**: an encrypted SQLite database with a CRDT-based write-ahead log.*
- $\mathcal{I}$ *is the **identity**: a DID document anchored on I-Chain, binding the capsule to one or more public keys.*
- Γ *is the **capability set**: a lattice of UCAN-style capability tokens governing read, write, sync, delegate, and revoke operations.*
- Σ *is the **sync topology**: the set of peers, relays, and chain anchors through which the vault replicates.*
- Π *is the **compute policy**: a specification of where computation may execute—local, provider-hosted, or confidential (FHE/TEE).*
- $\mathcal{A}$ *is the **audit chain**: an append-only sequence of anchored receipts on Lux recording policy changes, capability grants, and sync checkpoints.*
- $\mathcal{P}$ *is the **key policy**: the set of threshold, rotation, and recovery rules governing the capsule's cryptographic material, enforced on K-Chain.*

Definition 2.2 (Principal)**.** *A **principal** is any entity that owns a capsule: a human user, an organization, an AI agent, a device, or a shared workspace. Each principal has exactly one root capsule. Derived capsules (e.g., per-app or per-workspace vaults) inherit capabilities from the root through delegation chains in* Γ*.*

Definition 2.3 (Vault)**.** *The vault* $V = (\mathsf{DB}, \mathsf{WAL}, k_{\mathsf{enc}})$ *consists of:*

- DB*: a SQLite database encrypted with SQLCipher using AES-256-CBC with HMAC-SHA512 page authentication.*
- WAL*: a CRDT-based write-ahead log where each entry is a* $(\mathsf{hlc}, \mathsf{op}, \mathsf{sig})$ *triple—hybrid logical clock timestamp, operation, and signature from the writing principal.*
- k_{enc}*: the vault encryption key, derived from the principal's root key material via HKDF-SHA256, optionally wrapped under ML-KEM for post-quantum protection.*

2.1 Ownership Invariants

The Capsule model enforces four invariants that distinguish it from both traditional cloud architectures and fully on-chain systems.

Invariant 2.1 (Key Sovereignty)**.** *The vault encryption key* k_{enc} *is never transmitted in plaintext to any party other than the owning principal. Providers receive only ciphertext. Recovery requires threshold reconstruction on T-Chain, not provider cooperation.*

Invariant 2.2 (Data Locality)**.** *The authoritative copy of the vault state is the local replica. Remote replicas (provider-hosted, peer-hosted) hold only encrypted snapshots and CRDT operation logs. A principal can reconstruct the full vault from any replica plus the encryption key.*

Invariant 2.3 (Provider Replaceability)**.** *No provider holds state that the principal cannot obtain elsewhere. Sync topology* Σ *can be reconfigured to exclude any provider at any time without data loss. The protocol guarantees that switching providers requires only re-pointing the sync target—no data migration, no key re-wrapping (unless the principal chooses to rotate).*

Invariant 2.4 (Audit Completeness)**.** *Every policy change, capability grant, capability revocation, and key rotation produces a receipt anchored in* $\mathcal{A}$*. The audit chain is append-only and anchored to Lux. A third party can verify the complete history of a capsule's trust configuration without access to the vault contents.*

Theorem 2.1 (Capsule Sovereignty)**.** *Under Invariants 2.1–2.4, a capsule $\mathcal{C}$ is **sovereign**: the owning principal can, at any time, (a) decrypt and read all vault contents, (b) revoke all delegated capabilities, (c) migrate to a new provider set, and (d) verify the complete trust history—without cooperation from any provider or peer.*

Proof. By Invariant 2.1, the principal holds k_{enc} (or can reconstruct it via T-Chain threshold recovery without provider participation), so (a) holds. By Invariant 2.2, at least one encrypted replica is locally held, so the principal has data availability independent of providers. Capability revocation (b) is a K-Chain transaction signed by the principal's root key—no provider co-signature is required because Γ is a unilateral delegation lattice. Migration (c) follows from Invariant 2.3: the principal re-points Σ and the new provider receives the encrypted snapshot. Trust history verification (d) follows from Invariant 2.4: $\mathcal{A}$ is on Lux, readable by any party. □

2.2 Capsule Lifecycle

A capsule transitions through four phases:

1. **Genesis**: The principal generates root key material, derives k_{enc}, creates the initial encrypted vault, registers the DID on I-Chain, and publishes the initial key policy to K-Chain.
2. **Active**: The vault accepts local writes, syncs via Σ, and anchors checkpoints to $\mathcal{A}$. Capabilities in Γ may be delegated and revoked.
3. **Recovery**: If the principal loses access to k_{enc}, threshold reconstruction on T-Chain produces a new key share set. The vault is re-encrypted under the new key. The recovery event is recorded in $\mathcal{A}$.
4. **Termination**: The principal publishes a revocation of all capabilities and a tombstone DID document. Provider replicas are garbage-collected. The audit chain remains on Lux permanently.

3 Three Classes of Applications

The Capsule model supports three distinct application classes, each with different trust assumptions and cryptographic requirements.

3.1 Class I: Local-First Private Applications

The simplest class. The vault is the entire application backend. All reads and writes happen locally. Sync is peer-to-peer or through an encrypted relay. The chain is used only for identity and audit anchors.

Examples. Note-taking apps, personal finance trackers, password managers, local AI assistants, private journals, health records.

Trust model. The principal trusts only their own device and key material. Providers see ciphertext. The chain sees only anchor hashes and capability metadata.

Definition 3.1 (Class I Application)**.** *A Class I application is a pair $(f, \mathcal{C})$ where f is a deterministic function $f : V \times \mathsf{Input} \to V \times \mathsf{Output}$ that executes entirely on the principal's device, reading from and writing to the local vault V. Sync propagates encrypted CRDT operations; no provider or peer sees plaintext.*

Provider portability. A Class I app can switch sync providers by updating Σ. The new provider receives the encrypted CRDT log. No re-encryption is necessary because providers never held the key.

3.2 Class II: Threshold Applications

Applications requiring multi-party authorization. The vault may be shared across principals, with operations gated by threshold policies on K-Chain and secret reveals on T-Chain.

Examples. Treasury management (3-of-5 approval for transfers), organizational admin recovery, secrets management (reveal API key to authorized member), multi-party document signing, escrow release.

Trust model. No single principal controls the operation. A quorum of t-of-n participants must authorize via T-Chain. The chain enforces the threshold—providers cannot bypass it.

Definition 3.2 (Class II Application)**.** *A Class II application is a triple* $(g, \mathcal{C}_1, \ldots, \mathcal{C}_n)$ *where* g *is a function that executes only when* t*-of-*n *capsule owners produce valid authorization proofs on T-Chain. The function* g *may read from multiple vaults (with delegated capabilities) and write results back to designated vaults.*

Threshold mechanics. The authorization proof is a threshold signature aggregated on T-Chain using the protocol specified in [7]. Each participant signs a *proposal hash* containing the operation description, target vault identifiers, and an expiration timestamp. The chain verifies the threshold and emits an authorization receipt before g executes.

3.3 Class III: Confidential Compute Applications

Applications where computation must occur on encrypted data without any party (including the compute provider) seeing plaintext. This class uses fully homomorphic encryption (FHE) or trusted execution environments (TEEs).

Examples. Private AI inference (run a model on encrypted user data), encrypted analytics (aggregate statistics without seeing individual records), sealed-bid auctions, private credit scoring, confidential voting.

Trust model. The compute provider executes on ciphertext. Results are encrypted under the requesting principal's key. For FHE, trust is purely cryptographic. For TEE, trust additionally assumes hardware attestation integrity.

Definition 3.3 (Class III Application)**.** *A Class III application is a tuple* $(h, \mathcal{C}, \mathsf{CP})$ *where* h *is a function compiled to an FHE circuit or TEE enclave,* $\mathcal{C}$ *is the requesting capsule, and* CP *is a confidential compute provider. The provider evaluates* h *on* $\mathsf{Enc}(x)$ *where* x *is extracted from* V *and encrypted under a scheme supporting homomorphic evaluation. The result* $\mathsf{Enc}(h(x))$ *is returned to the principal for local decryption.*

FHE integration. Class III applications use the TFHE scheme specified in [8] for boolean circuits and CKKS for approximate arithmetic. Key generation and decryption happen inside the capsule. The FHE evaluation key (public) is sent to the compute provider. The secret key never leaves the vault.

3.4 Application Class Summary

Table 1: Comparison of application classes.

	Class I	**Class II**	**Class III**
Compute location	Local device	Local + chain	Provider (encrypted)
Chain usage	Anchor, identity	Threshold enforce	Compute verification
Provider sees	Ciphertext only	Ciphertext only	Ciphertext + eval key
Crypto required	AES + ML-KEM	+ threshold sigs	+ FHE or TEE
Latency	Sub-ms	Seconds (chain)	Seconds–minutes

4 The Trust Kernel

Lux serves as the trust kernel for all capsules. Trust is factored across four purpose-built chains, each optimized for a specific class of trust operation.

4.1 Architecture

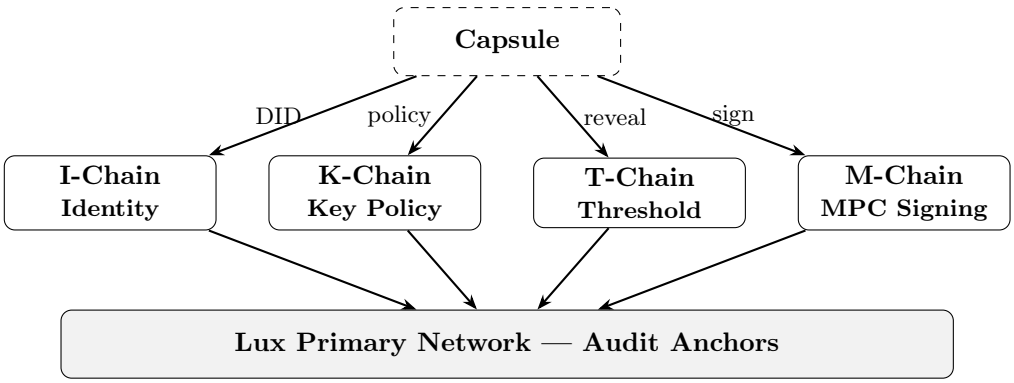

Figure 1: Trust kernel architecture. Four purpose-built chains anchor to the Lux primary network.

4.2 I-Chain: Identity

I-Chain stores DID documents conforming to the W3C DID Core specification. Each capsule has exactly one DID anchored on I-Chain. The DID document contains:

- Public keys (verification methods) for authentication and capability invocation.
- Service endpoints for sync relay discovery.
- Controller relationships for organizational hierarchies.
- Post-quantum key material (ML-DSA public keys) alongside classical Ed25519 keys for hybrid verification.

DID resolution is a read from I-Chain state. DID updates require a transaction signed by the current controller key. The full specification is in [11].

4.3 K-Chain: Key Policy

K-Chain stores and enforces key lifecycle policies:

- **Rotation schedules**: mandatory key rotation intervals, grace periods, and automatic revocation on expiry.
- **Threshold definitions**: (t, n) parameters for recovery, delegation, and administrative actions.

- **Wrapping policy**: which keys may wrap which other keys, preventing unauthorized key export.
- **Capability constraints**: maximum delegation depth, time-bounded grants, resource-scoped permissions.

K-Chain validators enforce policy at the transaction level. A key rotation transaction that violates the capsule's policy (e.g., rotating before the minimum interval) is rejected at consensus.

4.4 T-Chain: Threshold Reveal

T-Chain coordinates threshold operations: secret sharing, reconstruction, and time-locked reveals. The protocol:

1. A proposer submits a *reveal request* specifying the secret identifier, the threshold (t, n), and an expiration block height.
2. Each share holder submits their share as an encrypted payload (encrypted to the proposer's public key) within the expiration window.
3. Once t shares are submitted, T-Chain verifies the Feldman VSS commitments and emits a *reveal receipt*.
4. The proposer reconstructs the secret locally.

The chain never sees the secret in plaintext. It verifies share validity through commitment schemes and enforces the threshold policy defined on K-Chain.

4.5 M-Chain: MPC Signing

M-Chain provides decentralized multi-party computation for cross-chain signing, as specified in [7]. Capsules interact with M-Chain when they need to authorize operations on external chains (Bitcoin, Ethereum, etc.) without exposing a single private key.

The signing flow: the capsule submits a signing request to M-Chain. Validators holding key shares execute the threshold signature protocol (CGG21 for ECDSA, FROST for EdDSA, Pulsar for post-quantum). The aggregated signature is returned to the capsule and broadcast to the target chain.

5 The Data Plane

The data plane is Base: an encrypted SQLite runtime with CRDT sync, realtime subscriptions, and a plugin architecture.

5.1 Encrypted SQLite

Each vault is a single SQLite database file encrypted by SQLCipher. The encryption parameters:

- Algorithm: AES-256-CBC with per-page HMAC-SHA512.
- Key derivation: PBKDF2-HMAC-SHA512, 256,000 iterations for passphrase-derived keys. Direct key mode for HKDF-derived keys.
- Page size: 4096 bytes (matching OS page size for mmap efficiency).
- Reserve bytes: 80 (IV + HMAC per page).

The vault key k_{enc} is derived from the capsule's root key material:

$$k_{\mathsf{enc}} = \mathsf{HKDF\text{-}SHA256}(k_{\mathsf{root}}, \texttt{"vault"} \| \mathsf{capsule\_id}, 32)$$

For post-quantum key wrapping, k_{enc} is additionally encapsulated under ML-KEM-768:

$$(c, k_{\mathsf{shared}}) \leftarrow \mathsf{ML\text{-}KEM.Encaps}(pk_{\mathsf{pq}})$$

$$k_{\mathsf{wrapped}} = \mathsf{AES\text{-}256\text{-}GCM.Enc}(k_{\mathsf{shared}}, k_{\mathsf{enc}})$$

5.2 CRDT Sync Protocol

The sync protocol operates on the write-ahead log, not the database file. Each write produces a CRDT operation:

Definition 5.1 (Sync Operation)**.** *A sync operation is a tuple* $(\mathsf{hlc}, \mathsf{table}, \mathsf{row\_id}, \mathsf{col}, \mathsf{value}, \mathsf{tombstone}, \sigma)$ *where* hlc *is a hybrid logical clock timestamp, and* σ *is an Ed25519 signature over the preceding fields by the writing principal.*

Conflict resolution uses last-writer-wins (LWW) semantics ordered by HLC. For columns requiring richer merge semantics (e.g., collaborative text), the application may register custom CRDT types (RGA for sequences, OR-Set for collections).

The sync protocol:

1. The local vault writes an operation to the WAL and increments the HLC.
2. The operation is encrypted with k_{sync} (derived from k_{root} via a separate HKDF context) and sent to peers in Σ.
3. Receiving peers decrypt, verify σ, check capability authorization in Γ, and merge into their local WAL.
4. Periodically, the vault compacts the WAL into the SQLite database and publishes a checkpoint hash to $\mathcal{A}$.

5.3 Realtime API

Base exposes a realtime subscription API over the vault. Clients subscribe to table/row/column changes. The subscription is local—it observes the CRDT merge stream. When a remote peer's operation merges into the local WAL, subscribed clients receive the change event. No server is in the loop.

5.4 Plugin Runtime

Base supports server-side plugins for operations that cannot run purely client-side: scheduled tasks, webhook handlers, schema migrations, and background sync jobs. Plugins execute in a sandboxed runtime (V8 isolate or WASM) with read/write access scoped by Γ. A plugin cannot access vault data outside its granted capabilities.

6 The Post-Quantum Security Stack

The Capsule security stack is tiered by application class, with increasing cryptographic complexity as trust requirements grow.

6.1 Tier 1: Default (Class I)

All capsules use this tier. It protects vault data and sync traffic.

- **Vault encryption**: AES-256-CBC (SQLCipher) with HMAC-SHA512 page authentication.
- **Key encapsulation**: ML-KEM-768 (NIST FIPS 203) for wrapping vault keys, providing IND-CCA2 security against quantum adversaries.
- **Signatures**: Ed25519 for real-time operations (performance), ML-DSA-65 (FIPS 204) for policy transactions on-chain (quantum resistance).
- **Sync encryption**: AES-256-GCM for operation payloads in transit.
- **Key derivation**: HKDF-SHA256 from root key material.

Theorem 6.1 (Tier 1 IND-CPA Security)**.** *Under the Module-LWE assumption with parameters* $(n = 768, q = 3329, \eta = 2)$ *and the assumption that AES-256 is a secure pseudorandom permutation, a Tier 1 capsule vault is IND-CPA secure against a quantum polynomial-time adversary with access to encrypted vault snapshots and sync traffic.*

Proof sketch. The vault key k_{enc} is wrapped under ML-KEM-768, which is IND-CCA2 secure under Module-LWE [9]. An adversary with access to the ciphertext vault file and encrypted sync operations sees only AES-256-CBC ciphertext (vault) and AES-256-GCM ciphertext (sync). Without k_{enc} or k_{sync} (both derived from k_{root} and protected by ML-KEM), the adversary cannot distinguish vault contents from random, giving IND-CPA. The HMAC-SHA512 page authentication additionally provides integrity (IND-CCA under the encrypt-then-MAC composition theorem). □

6.2 Tier 2: Threshold Control (Class II)

Adds threshold cryptographic protocols for multi-party authorization.

- **Secret sharing**: Feldman VSS over a prime-order group for verifiable share distribution.
- **Threshold signatures**: FROST (EdDSA), CGG21 (ECDSA), Pulsar (post-quantum) as specified in [7].
- **Time-locks**: Verifiable delay functions (VDFs) for time-bounded secret reveals.

6.3 Tier 3: Confidential Compute (Class III)

Adds homomorphic encryption for computation on encrypted data.

- **Boolean circuits**: TFHE with bootstrapping for arbitrary boolean function evaluation [8].
- **Approximate arithmetic**: CKKS for floating-point operations (ML inference, statistical aggregation).
- **Hybrid FHE-MPC**: Combine FHE evaluation with MPC for multi-input confidential computation, as described in [10].
- **TEE fallback**: Intel TDX or ARM CCA for workloads where FHE latency is prohibitive, with remote attestation verified on-chain.

Table 2: Post-quantum security tiers.

Tier	Primitives	PQ Safe	Overhead
1 (Default)	AES-256, ML-KEM, ML-DSA	Yes	Negligible
2 (Threshold)	+ Feldman VSS, FROST, Pulsar	Yes	Seconds
3 (Confidential)	+ TFHE, CKKS	Yes (FHE)	10–1000×

7 Provider Model

Providers are participants in the Web5 ecosystem that offer services to capsule owners. They are not data owners. They never hold decryption keys. They are replaceable.

7.1 Provider Roles

Definition 7.1 (Provider)**.** *A provider is a registered entity on I-Chain offering one or more services from the set* $\{\mathsf{relay}, \mathsf{storage}, \mathsf{gateway}, \mathsf{recovery}, \mathsf{compute}, \mathsf{index}\}$*, with service quality commitments published on-chain and enforced by staking.*

- **Relay**: Forwards encrypted CRDT operations between peers. Sees ciphertext. Provides availability, not confidentiality.

- **Storage**: Hosts encrypted vault snapshots and WAL segments. Provides durability. Subject to proof-of-storage challenges.
- **Gateway**: HTTP/WebSocket endpoint for client applications. Handles authentication (DID-based), rate limiting, and protocol translation. Does not decrypt.
- **Recovery**: Holds encrypted key shares for threshold recovery. Cannot reconstruct alone (requires t-of-n on T-Chain).
- **Compute**: Executes FHE circuits or TEE enclaves for Class III applications. Receives evaluation keys and ciphertext. Returns encrypted results.
- **Index**: Maintains searchable indexes over encrypted metadata (using techniques like order-preserving encryption or searchable symmetric encryption) for query acceleration.

7.2 Provider Economics

Providers stake LUX on the primary network. Capsule owners pay providers in LUX for services consumed. Payments are per-operation (relay fees), per-byte-month (storage fees), or per-circuit-evaluation (compute fees). Service level agreements are on-chain; slashing occurs for downtime or data loss verified by proof-of-storage audits.

7.3 Provider Switching

Switching providers is a four-step process:

1. Update Σ to include the new provider.
2. The new provider receives the encrypted vault snapshot and catches up on the CRDT log from an existing peer or the old provider.
3. Verify replication by comparing checkpoint hashes.
4. Update Σ to exclude the old provider.

No key rotation is required. No data re-encryption is required. The principal does not need to be online during the transfer—any peer in Σ can serve the encrypted state.

8 Agent Capsules

AI agents are first-class principals in the Capsule model. An agent gets its own capsule with its own vault, identity, capabilities, and audit chain. This is not a metaphor. An agent capsule has the same formal structure as a human capsule.

8.1 Agent as Sovereign Process

Definition 8.1 (Agent Capsule)**.** *An **agent capsule** is a capsule $\mathcal{C}_a$ where the principal is an autonomous software process. The agent's vault stores its memory (conversation history, learned preferences, tool outputs), its capability set Γ_a defines what APIs, data sources, and actions it may access, and its key policy $\mathcal{P}_a$ defines who can inspect, modify, or terminate the agent.*

Key differences from human capsules:

- **Delegated authority**: An agent's root key is typically generated by a human principal and delegated via Γ with explicit scope and time bounds. The agent cannot escalate its own capabilities.
- **Budget constraints**: $\mathcal{P}_a$ includes spending limits— maximum LUX per transaction, per day, and per provider interaction. Exceeded budgets halt the agent until the delegating principal re-authorizes.
- **Auditable memory**: Every write to the agent's vault is signed and anchored. A human principal can audit the complete history of what the agent stored, when, and in response to what input.

- **Private inference**: For Class III agent operations, inference runs on encrypted data via FHE or inside a TEE. The compute provider sees neither the model weights nor the user's input.

8.2 Agent Capability Model

Agent capabilities are structured as a DAG of UCAN tokens:

```
human-root
  +-- agent-root (time-bounded, budget-limited)
        +-- read:vault/user-preferences
        +-- write:vault/agent-memory
        +-- call:api/weather (rate-limited)
        +-- call:api/calendar (read-only)
        +-- spend:lux/0.1-per-tx
```

Each capability token is a signed JWT with fields: `iss` (delegator DID), `aud` (agent DID), `att` (resource + action), `exp` (expiration), `prf` (proof chain back to root). Revocation is a K-Chain transaction.

8.3 Agent Lifecycle

1. **Spawn**: Human principal creates agent capsule, generates agent keys, delegates capabilities, sets budget.
2. **Execute**: Agent reads from its vault, invokes tools within Γ_a, writes results to its vault, anchors receipts.
3. **Inspect**: Human principal reads agent vault (using delegated read capability on $\mathcal{C}_a$) to review memory and actions.
4. **Revoke**: Human principal revokes Γ_a on K-Chain. Agent can no longer invoke tools or write to its vault. Vault contents remain readable for audit.

9 Migration Path

Existing applications built on Firebase, Supabase, Notion APIs, or similar centralized backends can migrate to capsule-based architectures incrementally.

9.1 Migration Mapping

Table 3: Migration from centralized backends to capsule architecture.

Centralized	Capsule Equivalent	Trust Model
Firebase Auth	I-Chain DID + session tokens	Self-sovereign
Firestore	Encrypted SQLite vault	Local-first
Realtime DB	CRDT sync over Σ	Peer-to-peer
Cloud Storage	Encrypted blob at provider	Provider-blind
Cloud Functions	Base plugins (WASM/V8)	Sandboxed
Security Rules	Γ capability lattice	Cryptographic
Push notifications	Sync event subscriptions	Local observer
Analytics	Aggregation via Class III FHE	Confidential

9.2 Migration Strategy

1. **Phase 1: Dual-write.** The application writes to both the existing backend and a local capsule vault. Reads come from the centralized backend. This validates data model compatibility.
2. **Phase 2: Local-read.** Reads switch to the local vault. The centralized backend becomes a sync target (write-through). The user experiences offline capability and reduced latency.
3. **Phase 3: Cut-over.** The centralized backend is removed from Σ. The capsule's CRDT sync handles replication. Identity migrates to I-Chain. The old backend is decommissioned.

The key constraint: the application's data model must fit SQLite's relational schema. Document databases (Firestore, MongoDB) require schema normalization. Key-value stores map directly. Graph data requires an adjacency-list representation or a dedicated CRDT type.

10 Hardest Unsolved Problems

Honesty demands listing what we have not solved. These are open research problems, not engineering tasks.

10.1 Key Rotation UX

Rotating a capsule's root key requires re-wrapping k_{enc} and updating the DID document. For a single-device user, this is straightforward. For a user with 10 devices, 5 shared workspaces, and 3 agent capsules, key rotation becomes a coordination problem. Every derived key must be re-derived. Every peer in Σ must receive the new sync key. Every agent's delegated capabilities must be re-issued.

The current protocol handles this through a rotation transaction on K-Chain that emits a rotation event, triggering re-keying across all derived capsules. The UX problem: if any device is offline during rotation, it holds stale keys and cannot sync until it comes online and processes the rotation event. We do not have a satisfying solution for the offline-device-during-rotation case beyond "queue the rotation and apply on reconnect," which creates a window where the old key is still valid.

10.2 Partial Sharing

A principal wants to share rows 1–50 of a table but not rows 51–100. The current capability model operates at table granularity. Row-level sharing requires either: (a) splitting the table into separate vaults (one shared, one private), which fragments the schema; or (b) implementing row-level encryption with per-row keys, which multiplies key management complexity.

Neither approach is clean. The fundamental tension: SQLite operates on pages, not rows. Encrypting at row granularity means abandoning SQLCipher's page-level encryption for a custom scheme with worse performance characteristics.

10.3 Sync Metadata Leakage

Even with encrypted payloads, the sync protocol leaks metadata: which capsules sync with which peers, how often, and how much data flows. A traffic analysis adversary can infer social graphs and activity patterns from sync metadata alone.

Mitigations include constant-rate padding (expensive), mix networks for relay traffic (high latency), and PIR for checkpoint retrieval (computationally expensive). None are satisfactory for real-time sync workloads.

10.4 Private Search and Indexing

Searching encrypted vault data is an open problem. Searchable symmetric encryption (SSE) leaks access patterns. Oblivious RAM (ORAM) has $O(\log n)$ overhead per access. FHE-based search is orders of magnitude slower than plaintext search.

For Class I applications, search is local and fast (plaintext after decryption). The problem arises for Class II and III applications where a provider must search on behalf of a principal without seeing plaintext. Practical private search at scale remains unsolved.

10.5 Agent Memory Permissions

An agent's vault accumulates memory over time: conversation history, learned preferences, tool outputs. The principal who spawned the agent should be able to: (a) read all agent memory, (b) selectively delete memories, (c) prevent the agent from remembering certain interactions.

The challenge: if the agent has write access to its own vault (necessary for it to function), it can write anything, including data the principal wishes it would not retain. Enforcing "do not remember this" requires either trusted hardware (TEE for the agent runtime) or a formal verification framework for agent write policies. Neither is production-ready.

10.6 Cross-Application Composition

A user wants Application A's data to feed into Application B's computation. Both applications use separate capsules. The composition requires: (a) a capability delegation from Capsule A to Capsule B, (b) a data format agreement, and (c) a sync path between the two vaults.

The protocol supports this mechanically (delegate a read capability, establish a sync link), but the UX is unresolved. Users should not need to understand capability lattices and CRDT topologies to connect two apps. The equivalent of "Sign in with Google" for cross-app data sharing in Web5 does not exist yet.

11 Related Work

Solid (Berners-Lee). Solid [2] proposed personal data pods where users store their data and grant applications access. The Capsule model shares Solid's vision of data sovereignty but differs in three ways: (1) Capsule vaults are encrypted at rest (Solid pods are not); (2) sync is CRDT-based rather than LDP-based; (3) trust is anchored to a blockchain rather than relying on WebID-TLS.

Fission (UCAN). Fission [3] introduced UCAN (User Controlled Authorization Networks) for decentralized capability delegation. The Capsule capability set Γ builds directly on UCAN semantics. The difference: Capsule capabilities are enforced on K-Chain, providing revocation guarantees that pure UCAN (off-chain) cannot offer.

Ceramic. Ceramic [4] provides mutable data streams anchored to a blockchain. Capsules differ by storing data locally (not on IPFS), using SQLite (not JSON streams), and supporting offline-first operation without network access.

DXOS. DXOS [5] builds local-first collaborative applications with ECHO (Eventually Consistent Hierarchical Objects). The Capsule CRDT sync model is closest to DXOS in spirit. The distinction: Capsules add a formal trust kernel (Lux) for identity, key policy, and threshold operations that DXOS handles through simpler peer-to-peer key exchange.

Local-First Software. Kleppmann et al. [1] articulated seven ideals for local-first software: fast, multi-device, offline, collaborative, long-lived, private, and user-owned. The Capsule model is a concrete architecture that satisfies all seven, adding the trust kernel that the original manifesto identified as missing.

Evervault. Evervault [6] provides encryption-as-a-service with relay proxies that encrypt data before it reaches application servers. Capsules subsume this pattern: the vault is always encrypted, and providers never see plaintext. The difference is architectural—Evervault is a middlebox; Capsules are a runtime.

12 Conclusion

The Capsule is not a decentralized database. It is not a blockchain wallet. It is not a privacy layer bolted onto an existing system.

The Capsule is a sovereign encrypted process. It is the unit of tenancy, the unit of privacy, the unit of collaboration, and the unit of computation in Web5. Every principal—human, organizational, algorithmic—gets one. The vault holds state. The identity anchors trust. The capability set governs access. The sync topology handles replication. The compute policy determines where code runs. The audit chain proves what happened.

Base owns state: encrypted SQLite, CRDT sync, realtime APIs, plugin runtime. Lux owns trust: identity on I-Chain, key policy on K-Chain, threshold reveals on T-Chain, MPC signing on M-Chain, audit anchors on the primary network. Providers own service quality: relay throughput, storage durability, compute availability. Users own keys and exit rights: they can decrypt, revoke, migrate, and audit at any time, without permission from any provider.

This is what it means for the personal computer to return—not as a device on a desk, but as a cryptographic process that follows you across every device, every application, and every provider. The Capsule is the personal computer of Web5.

The hard problems remain hard. Key rotation UX is unsolved for multi-device principals. Partial sharing requires schema-level compromises. Sync metadata leaks to traffic analysis. Private search at scale is open. Agent memory governance needs formal verification. Cross-app composition lacks a usable consent flow.

We have listed these problems explicitly because solving them is the work that matters. The architecture is sound. The cryptography is proven. The chains are specified. What remains is the engineering and UX research to make sovereignty invisible—to make it so natural that users never think about keys, capabilities, or sync topologies, and yet own everything.

References

[1] M. Kleppmann, A. Wiggins, P. van Hardenberg, and M. McGranaghan, "Local-First Software: You Own Your Data, in Spite of the Cloud," *Onward! 2019*, ACM, 2019.

[2] T. Berners-Lee and D. Sambra, "Solid: A Platform for Decentralized Social Applications Based on Linked Data," MIT CSAIL Technical Report, 2016.

[3] B. Holmgren and P. Chiusano, "UCAN: User Controlled Authorization Networks," Fission Specification, 2021.

[4] M. Zargham, J. Clark, and D. Buchner, "Ceramic: A Decentralized Data Network for Web3 Applications," 3Box Labs, 2020.

[5] R. Brinkman et al., "DXOS: A Framework for Local-First Collaborative Applications," DXOS Technical Report, 2023.

[6] S. Agarwal and C. Sheridan, "Evervault: Encryption as Infrastructure," Evervault Whitepaper, 2022.

[7] Z. Kelling and V. Seesahai, "M-Chain: Decentralized Multi-Party Computation Custody with Quantum-Safe Threshold Signatures," Lux Industries, 2023.

[8] Z. Kelling, "Torus Threshold Fully Homomorphic Encryption: Boolean Circuits on Encrypted Data with Distributed Decryption," Lux Industries, 2025.

[9] Z. Kelling, "Post-Quantum Cryptographic Suite for EVM: ML-KEM, ML-DSA, and SLH-DSA as Native Precompiles," Lux Industries, 2024.

[10] Z. Kelling, "Hybrid FHE-MPC Protocols for Multi-Input Confidential Computation on Lux," Lux Industries, 2025.

[11] Z. Kelling, "Lux ID: Decentralized Identifier Specification for Multi-Chain Identity," Lux Industries, 2024.

[12] Z. Kelling, "Lux Tokenomics: Economic Design for Multi-Chain Consensus Networks," Lux Industries, 2023.

[13] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-Free Replicated Data Types," *SSS 2011*, LNCS 6976, Springer, 2011.

[14] National Institute of Standards and Technology, "Module-Lattice-Based Key-Encapsulation Mechanism Standard," FIPS 203, 2024.

[15] National Institute of Standards and Technology, "Module-Lattice-Based Digital Signature Standard," FIPS 204, 2024.

[16] R. Hipp, "SQLite: Past, Present, and Future," *Proc. VLDB Endow.*, 15(12), 2022.