Lux as Trust Kernel: Identity, Key Policy, Threshold Governance, and Receipts

Zach Kelling

Lux Industries

2025

Abstract

We formalize Lux Network as a *trust kernel*: a minimal on-chain substrate that provides identity binding, key lifecycle policy, threshold governance, and tamper-evident receipts—without storing application data on-chain. The trust kernel model inverts the common blockchain-as-database pattern. Applications retain full data sovereignty in local encrypted vaults (capsules) while delegating exactly four concerns to the chain: (i) globally-unique identity via decentralized identifiers (`did:lux:*`), (ii) key registration and rotation policy on the K-Chain, (iii) quorum-controlled reveal and treasury approval on the T-Chain with TFHE policy evaluation and CKKS confidential compute, and (iv) multi-party threshold signatures on the M-Chain via FROST and CGGMP21. Every state transition produces a cryptographic receipt anchored on-chain, enabling provider-independent auditability. We define the trust kernel formally, prove that it preserves capsule sovereignty under Byzantine fault tolerance assumptions, and show that the architecture enables cross-capsule coordination (key exchange, capability delegation, threshold approval) without revealing plaintext to any single party. All critical protocols are verified in Lean 4 against mechanized proofs for consensus safety, FROST unforgeability, CGGMP21 threshold composition, and trust authority delegation.

Keywords: trust kernel, decentralized identity, threshold governance, key policy, receipt anchoring, post-quantum cryptography, TFHE, CKKS, FROST, CGGMP21

Contents

1 Introduction

Blockchains are routinely misapplied as general-purpose databases. This conflation creates systems that are simultaneously too expensive for data storage and too opaque for trust verification. The root cause is a failure to distinguish between two orthogonal concerns: *data management* (storage, indexing, query) and *trust management* (identity, authorization, coordination, audit).

Definition 1.1 (Trust Kernel)**.** *A trust kernel $\mathcal{K}$ is a replicated state machine that provides exactly four services to application capsules:*

(a) ***Identity****: globally-unique, self-sovereign identifiers bound to cryptographic key material;*

(b) ***Key Policy****: lifecycle management (registration, rotation, revocation) with enforceable policy constraints;*

(c) ***Threshold Governance****: quorum-controlled state transitions requiring t-of-n authorization;*

(d) ***Receipts****: tamper-evident, append-only records of every trust-relevant state transition.*

The trust kernel stores no application data. It stores *commitments to decisions about data*: who created it, who may access it, under what conditions, and when those conditions were evaluated. This separation is not merely architectural preference—it is a security boundary. Data that never touches the chain cannot leak from the chain.

1.1 What Local Vaults Cannot Provide

An encrypted local vault (a capsule, as defined in [1]) provides strong confidentiality and integrity for data at rest. However, a vault alone cannot provide:

(i) **Global identity**: a vault can generate keypairs, but cannot prove uniqueness or non-repudiation to a third party without a shared root of trust;

(ii) **Cross-party coordination**: two vaults cannot agree on a shared secret, delegate capabilities, or approve a joint action without a coordination layer that both trust;

(iii) **Settlement finality**: a vault can record local state, but cannot prove to an external verifier that a state transition occurred at a specific time and was not subsequently reversed.

The trust kernel fills exactly these gaps. It is the minimum viable chain: the smallest on-chain footprint that makes sovereign applications composable.

1.2 Contribution

This paper makes the following contributions:

1. We formalize the trust kernel abstraction and prove it sufficient for sovereign application composition (Section 1).
2. We specify the K-Chain key registry with ML-KEM wrapping, rotation policy, and revocation receipts (Section 2).
3. We define T-Chain threshold governance with TFHE policy evaluation and CKKS confidential compute (Section 3).
4. We describe M-Chain MPC signing via FROST and CGGMP21 (Section 4).

5. We specify the I-Chain decentralized identity layer with ZK selective disclosure (Section 5).
6. We define the receipt anchoring protocol and prove its tamper-evidence (Section 6).
7. We show how the trust kernel enables provider switching without data loss (Section 7).
8. We formalize cross-capsule coordination through the trust kernel (Section 8).
9. We present the security model with formal trust assumptions and PQ migration path (Section 9).
10. We reference mechanized Lean 4 proofs for all critical protocols (Section 10).

2 K-Chain: Key Registry and Policy

The K-Chain is a purpose-built chain within the Lux primary network that manages the lifecycle of cryptographic keys. It does not store encrypted data; it stores *policy about keys*.

2.1 Key Registry

Definition 2.1 (Key Record)**.** *A key record κ is a tuple:*

$$\kappa = (\mathsf{id}, \mathsf{pk}, \mathsf{alg}, \mathsf{caps}, \mathsf{policy}, \mathsf{created}, \mathsf{expires}, \mathsf{status})$$

where $\mathsf{id} \in \{0,1\}^{256}$ *is the key identifier,* pk *is the public key,* $\mathsf{alg} \in \{$*ML-KEM-768*, *ML-KEM-1024*, *Ed25519*, *ML-*
is the algorithm identifier, caps *is a capability bitfield,* policy *is a policy predicate,* created *and* expires *are block heights, and* $\mathsf{status} \in \{$*ACTIVE*, *ROTATED*, *REVOKED*$\}$.

2.2 ML-KEM Wrapping

Keys stored in capsule vaults are wrapped using ML-KEM (FIPS 203) as specified in [4]. The K-Chain stores only the public encapsulation key ek and the policy governing who may request encapsulation. The wrapped key material never appears on-chain.

Definition 2.2 (Wrapping Policy)**.** *A wrapping policy π_w for key record κ is a predicate over requestor identity and context:*

$$\pi_w : \mathsf{DID} \times \mathsf{Context} \to \{\top, \bot\}$$

where Context *includes the requestor's capability set, the current block height, and an optional threshold requirement.*

When a capsule requires a wrapped key, it submits a WrapRequest transaction to the K-Chain. The chain evaluates π_w against the requestor's DID and, if satisfied, emits a WrapGrant receipt containing the ML-KEM ciphertext $c = \mathsf{Encaps}(\mathsf{ek}; r)$ and the shared secret K is delivered to the requestor off-chain via a secure channel.

2.3 Rotation Policy

Definition 2.3 (Rotation Rule)**.** *A rotation rule ρ for key κ specifies:*

$$\rho = (\mathsf{max\_age}, \mathsf{successor\_alg}, \mathsf{approval\_threshold}, \mathsf{grace\_period})$$

The K-Chain enforces that no key remains active beyond max_age *blocks past κ.*created*. Upon rotation, the old key enters* ROTATED *status and a new key record is created with algorithm* successor_alg.

Rotation may be triggered by three events: (1) age expiry, (2) explicit owner request, or (3) threshold governance vote on the T-Chain. In cases (1) and (3), the rotation is mandatory and produces a RotationReceipt regardless of owner action.

2.4 Capability Grants and Revocation

Capabilities are delegated via GrantCap transactions that append to the key record's capability set. Each grant is scoped to a specific DID, a capability type (read, sign, admin), and an optional time bound.

Revocation is immediate and final. A RevokeCap transaction sets the capability's status to revoked and emits a RevocationReceipt. The receipt includes the revoked capability hash, the revoking authority's DID, and the block height, providing a permanent audit trail.

Proposition 2.1 (Revocation Finality)**.** *Once a RevocationReceipt is finalized on the K-Chain, no subsequent transaction can restore the revoked capability for the same key–DID pair.*

3 T-Chain: Threshold Governance

The T-Chain handles quorum-controlled operations: actions that require t-of-n authorization before execution. Unlike simple multi-signature schemes, the T-Chain supports *policy evaluation over encrypted inputs* via TFHE and *confidential aggregation* via CKKS.

3.1 Quorum-Controlled Reveal

Definition 3.1 (Quorum Policy)**.** *A quorum policy Q is a tuple:*

$$Q = (n, t, \mathsf{voters}, \mathsf{predicate}, \mathsf{timeout})$$

where n is the total number of authorized voters, t is the threshold, $\mathsf{voters} \subseteq \mathsf{DID}^n$ *is the voter set,* predicate *is a boolean circuit evaluated on voter inputs, and* timeout *is a block height deadline.*

Applications of quorum-controlled reveal include:

- **Treasury approval**: t-of-n board members must approve a disbursement before the transaction is signed on M-Chain.
- **Admin recovery**: a compromised admin key can be rotated if t recovery contacts authorize the rotation on K-Chain.
- **Conditional access**: a document is revealed to a requestor only if t data stewards approve the access request.

3.2 TFHE Policy Evaluation

For policies where voter inputs must remain private (e.g., salary-based approval thresholds, credit scoring), the T-Chain evaluates the quorum predicate as a boolean circuit over TFHE-encrypted bits. Each voter submits an encrypted vote $\hat{v}_i = \mathsf{TFHE.Enc}(v_i)$. The chain evaluates the predicate circuit homomorphically:

$$\hat{r} = \mathsf{Circuit}(\hat{v}_1, \ldots, \hat{v}_n)$$

and the result is threshold-decrypted by the T-Chain validators, yielding $r \in \{0, 1\}$ without revealing individual votes. The TFHE parameters follow [5]: $N = 1024$, $q \approx 2^{27}$, 128-bit security.

3.3 CKKS Confidential Compute

For policies requiring arithmetic over real-valued inputs (e.g., weighted voting, aggregate risk scores), the T-Chain supports CKKS approximate homomorphic encryption. Each party submits $\hat{x}_i = \mathsf{CKKS.Enc}(x_i)$ where $x_i \in \mathbb{R}$. The chain computes:

$$\hat{y} = \sum_{i=1}^{n} w_i \cdot \hat{x}_i$$

and compares the result against a threshold τ. The aggregated value y is revealed; individual inputs x_i are not.

Proposition 3.1 (Vote Privacy)**.** *Under the LWE assumption with parameters (N, q, σ) as specified in [4], no coalition of fewer than t T-Chain validators can recover any individual vote v_i from the transcript of a quorum evaluation.*

3.4 Treasury Approval and Admin Recovery

Treasury operations bind T-Chain governance to M-Chain signing. A disbursement follows a strict sequence:

1. A proposer submits a TreasuryProposal to the T-Chain with the target address, amount, and justification hash.
2. Authorized voters submit encrypted approvals.
3. The T-Chain evaluates the quorum predicate.
4. If approved, the T-Chain emits a GovernanceReceipt that authorizes the M-Chain to produce a threshold signature on the disbursement transaction.
5. The M-Chain produces the signature only upon verifying the receipt.

Admin recovery follows the same pattern with the rotation request replacing the disbursement transaction. The recovery quorum is stored on the K-Chain as part of the key record's policy (Definition 2.3).

4 M-Chain: MPC Signing

The M-Chain provides threshold signature generation as a chain-native service. It implements two protocols: FROST for Schnorr-based curves (Ed25519, BIP-340) and CGGMP21 for ECDSA (secp256k1). The full specification is given in [2]; here we describe the trust kernel interface.

4.1 FROST Threshold Signatures

Definition 4.1 (FROST Signing Session)**.** *A FROST signing session $\mathcal{S}$ for message m under group public key gpk with threshold t proceeds in two rounds:*

1. ***Commitment**: each signer i generates nonce pair $(d_i, e_i) \leftarrow \mathbb{Z}_q^2$ and broadcasts commitments $(D_i = g^{d_i}, E_i = g^{e_i})$.*
2. ***Response**: each signer i computes partial signature $z_i = d_i + \rho_i \cdot e_i + \lambda_i \cdot s_i \cdot c$ where $c = H(\mathsf{gpk}, R, m)$, ρ_i is the binding factor, and λ_i is the Lagrange coefficient.*

The aggregator computes $\sigma = (R, z)$ where $z = \sum_{i \in S} z_i$ and S is the signer set with $|S| \geq t$.

4.2 CGGMP21 Threshold ECDSA

For secp256k1 operations (Ethereum, Bitcoin legacy), the M-Chain uses the CGGMP21 protocol [2] which provides:

- Pre-signing phase with Paillier commitments for multiplicative-to-additive share conversion.
- Identifiable abort: if a signer submits an invalid partial signature, the protocol identifies and slashes the faulty party.
- Non-interactive online phase after a single pre-signing round.

4.3 Authorization Gate

The M-Chain will not produce a signature without a valid authorization. Authorization sources are:

1. A GovernanceReceipt from the T-Chain (for treasury operations).
2. A WrapGrant from the K-Chain (for key operations).
3. A valid DID authentication proof from the I-Chain (for user-initiated transactions).

Theorem 4.1 (Signing Authorization)**.** *Let σ be a threshold signature produced by the M-Chain. Then there exists a finalized receipt $\mathcal{R}$ on at least one of {K-Chain, T-Chain, I-Chain} that authorized the signing session. No valid signature can be produced without such a receipt.*

Proof. The M-Chain VM accepts SignRequest transactions only if accompanied by a receipt reference $(chain, block, txIndex)$. The VM verifies the receipt via Warp messaging [6] before initiating the signing protocol. A SignRequest without a valid receipt reference is rejected at the transaction validation layer, before entering the mempool. Since the signing protocol requires the SignRequest to be finalized on the M-Chain, and finalization requires valid receipt verification, the theorem holds. □

5 I-Chain: Decentralized Identity

The I-Chain implements the `did:lux:*` method as specified in [3]. Within the trust kernel, the I-Chain serves as the root of all identity assertions.

5.1 DID Document Structure

Definition 5.1 (Lux DID)**.** *A Lux DID has the form `did:lux:<method-specific-id>` where the method-specific identifier is derived from the subject's initial public key:*

$$\mathsf{msid} = \mathsf{Base58}(\mathsf{RIPEMD160}(\mathsf{SHA256}(\mathsf{pk}_0)))$$

The DID document contains verification methods (public keys), authentication relationships, service endpoints, and a controller field.

Each DID document is registered on the I-Chain as an immutable record. Updates produce new versions linked to the previous version by hash, forming a verifiable history chain.

5.2 Verifiable Credentials

The I-Chain supports W3C Verifiable Credentials with Lux-specific extensions:

- **Credential issuance**: an issuer signs a credential binding a claim to a subject DID and registers the credential hash on the I-Chain.
- **Credential verification**: a verifier checks the issuer's signature and confirms the credential hash exists on-chain with status ACTIVE.
- **Credential revocation**: the issuer submits a RevokeCredential transaction; the credential's on-chain status changes to REVOKED.

5.3 ZK Selective Disclosure

Credentials may be presented with zero-knowledge selective disclosure. The holder generates a ZK proof that demonstrates possession of a credential satisfying a predicate (e.g., "age $\geq$ 18") without revealing the full credential.

Definition 5.2 (Selective Disclosure Proof)**.** *Given a credential C with attributes $\{a_1, \ldots, a_k\}$ and a predicate ϕ over a subset of attributes, the holder produces a proof π such that:*

$$\mathsf{Verify}(\pi, \phi, \mathsf{issuer\_pk}, \mathsf{holder\_did}) = \top \iff \phi(a_{i_1}, \ldots, a_{i_m}) = \top$$

where $\{i_1, \ldots, i_m\} \subseteq \{1, \ldots, k\}$ and no information about attributes outside the predicate is revealed.

5.4 Credential Revocation via Accumulator

The I-Chain maintains a cryptographic accumulator for credential revocation. Adding a credential hash to the revocation accumulator constitutes revocation. Verification requires a non-membership witness: the holder proves their credential is *not* in the revocation set without downloading the full revocation list.

6 Receipt Anchoring

Every trust-relevant state transition in the Lux trust kernel produces a *receipt*: a compact, self-contained proof that the transition occurred.

Definition 6.1 (Trust Receipt)**.** *A trust receipt $\mathcal{R}$ is a tuple:*

$$\mathcal{R} = (\mathsf{chain}, \mathsf{block}, \mathsf{txHash}, \mathsf{type}, \mathsf{subject}, \mathsf{digest}, \mathsf{sig})$$

where chain $\in \{K, T, M, I\}$ *identifies the originating chain,* block *and* txHash *identify the transaction,* type *is the receipt type (Table 1),* subject *is the affected DID,* digest *is the SHA-256 hash of the receipt payload, and* sig *is a BLS aggregate signature from the chain's validator set.*

6.1 Capsule Audit Trails

A capsule maintains a local log of operations (encrypt, decrypt, share, revoke). For each operation, the capsule computes a digest and submits it as an AnchorTx to the appropriate chain. The chain includes the digest in a receipt. The capsule stores the receipt locally alongside the operation log entry.

Table 1: Trust receipt types by originating chain.

Chain	Receipt Type	Meaning
K	KeyRegistered	New key record created
K	KeyRotated	Key rotated to successor
K	CapGranted	Capability delegated
K	CapRevoked	Capability revoked
T	QuorumApproved	Threshold vote passed
T	QuorumRejected	Threshold vote failed
M	SignatureProduced	Threshold signature emitted
I	DIDRegistered	DID document created
I	CredentialIssued	Credential hash anchored
I	CredentialRevoked	Credential revoked

Theorem 6.1 (Audit Completeness)**.** *If a capsule faithfully anchors every operation and the trust kernel satisfies BFT safety (Section 9), then for any operation o in the capsule's log, there exists a finalized receipt $\mathcal{R}$ on-chain such that $\mathcal{R}.\mathsf{digest} = H(o)$.*

The on-chain receipt stores only the digest—never the operation payload. An auditor can verify completeness by comparing the capsule's local log against the receipt chain, without the chain ever learning the contents of the operations.

7 Provider Accountability

The trust kernel enables a critical property: *provider-independent auditability*. A capsule's data may be hosted by any provider (cloud, on-premise, edge), but the trust trail remains on-chain regardless of provider.

7.1 Provider Switching

When a capsule migrates from provider P_1 to provider P_2:

1. The capsule owner initiates a key rotation on the K-Chain, generating new wrapping keys accessible only to P_2.
2. P_1 produces a TransferReceipt attesting to the data handoff, anchored on-chain.
3. P_2 confirms receipt of the encrypted data and anchors a ReceiveReceipt.
4. The owner revokes P_1's capabilities on the K-Chain.

At no point does the chain store the capsule's data. The chain stores only the trust decisions (key rotation, capability revocation, transfer attestation) that make the migration auditable.

Proposition 7.1 (Provider Independence)**.** *Given a capsule $\mathcal{C}$ with receipt chain $\{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$ and a provider migration from P_1 to P_2, the receipt chain after migration contains sufficient information for any third-party auditor to verify: (a) the identity of both providers, (b) the time of migration, (c) the completeness of capability revocation from P_1, and (d) the continuity of the capsule's key lineage.*

8 Cross-Capsule Coordination

Two capsules $\mathcal{C}_A$ and $\mathcal{C}_B$ interact through the trust kernel without either capsule exposing plaintext to the other or to the chain.

8.1 Key Exchange

Capsule $\mathcal{C}_A$ retrieves $\mathcal{C}_B$'s public encapsulation key ek_B from the K-Chain, verifies it is active and satisfies $\mathcal{C}_A$'s trust policy, and performs ML-KEM encapsulation:

$$(c, K) \leftarrow \mathsf{ML\text{-}KEM.Encaps}(\mathsf{ek}_B)$$

The ciphertext c is delivered to $\mathcal{C}_B$ off-chain. The K-Chain records a CapGranted receipt noting that $\mathcal{C}_A$ was granted encapsulation rights for $\mathcal{C}_B$'s key. The shared secret K is used to derive a symmetric session key for subsequent encrypted communication.

8.2 Capability Delegation

$\mathcal{C}_A$ may delegate a capability to $\mathcal{C}_B$ by submitting a GrantCap transaction to the K-Chain:

$$\textsc{GrantCap}(\mathsf{key}_A, \mathsf{did}_B, \mathsf{cap}, \mathsf{expiry})$$

This creates a verifiable, time-bounded delegation chain. $\mathcal{C}_B$ can prove to any verifier that it holds capability cap over key_A by presenting the CapGranted receipt.

8.3 Threshold Approval

For operations requiring mutual consent (e.g., a joint contract signing), $\mathcal{C}_A$ and $\mathcal{C}_B$ create a 2-of-2 quorum on the T-Chain:

$$Q = (2, 2, \{\mathsf{did}_A, \mathsf{did}_B\}, \mathsf{AND}, \mathsf{timeout})$$

Both capsules submit their encrypted votes. The T-Chain evaluates the AND predicate. If both approve, a GovernanceReceipt is emitted that authorizes the subsequent action (e.g., M-Chain signing, K-Chain rotation).

Theorem 8.1 (Cross-Capsule Confidentiality)**.** *In the cross-capsule coordination protocol, the trust kernel learns: (i) the identities of the participating capsules, (ii) the type of coordination (key exchange, delegation, threshold approval), and (iii) the boolean outcome. The trust kernel does not learn the plaintext of any data exchanged between capsules, any attribute values used in ZK proofs, or any vote values in TFHE-encrypted quorum evaluations.*

9 Security Model

9.1 Trust Assumptions

The trust kernel's security rests on the following assumptions:

1. **BFT consensus**: each chain (K, T, M, I) is secured by a validator set V with $|V| = n$ and the adversary controls at most $f < n/3$ validators. Under this assumption, the chain provides safety (no conflicting receipts) and liveness (receipts are eventually finalized).

2. **Threshold honesty**: for t-of-n protocols, at least t participants are honest. This is strictly weaker than the BFT assumption when $t \leq \lfloor n/3 \rfloor + 1$.

3. **Cryptographic hardness**: ML-KEM is IND-CCA2 secure under the Module-LWE assumption; ML-DSA is EUF-CMA secure under Module-LWE and Module-SIS; TFHE is IND-CPA secure under LWE; FROST is EUF-CMA secure in the random oracle model under the discrete logarithm assumption; CGGMP21 is EUF-CMA secure under the strong RSA and DDH assumptions.

4. **Capsule integrity**: each capsule correctly implements its local encryption and anchoring protocols. A compromised capsule can leak its own data but cannot forge receipts or compromise other capsules through the trust kernel.

9.2 BFT Threshold Analysis

Theorem 9.1 (Trust Kernel Safety)**.** *If fewer than $n/3$ validators on each chain are Byzantine, then for any two receipts $\mathcal{R}_1, \mathcal{R}_2$ with the same* $(\mathsf{chain}, \mathsf{block}, \mathsf{txHash})$*, it holds that $\mathcal{R}_1 = \mathcal{R}_2$.*

Proof. This follows directly from the BFT safety property of the underlying consensus protocol. Two conflicting receipts at the same position would require two conflicting blocks at the same height, which the BFT consensus prevents when $f < n/3$. See the mechanized proof in `Consensus/BFT.lean` (Section 10). □

9.3 Post-Quantum Migration Path

The trust kernel is designed for incremental PQ migration:

1. **Phase 1 (current)**: ML-KEM for key wrapping on K-Chain; classical signatures (Ed25519, secp256k1) for transaction signing. All wrapped key material is PQ-safe at rest.
2. **Phase 2**: ML-DSA for transaction signatures on K-Chain and I-Chain. T-Chain and M-Chain retain classical signatures for threshold protocol compatibility.
3. **Phase 3**: Pulsar (lattice-based threshold signatures) replaces FROST on M-Chain, providing full PQ security for threshold operations.

Each phase is activated by a governance vote on the T-Chain. The K-Chain's rotation policy (Definition 2.3) includes the $\mathsf{successor\_alg}$ field, enabling per-key migration.

10 Formal Verification

All critical protocols in the trust kernel have corresponding Lean 4 mechanized proofs. The proof modules are organized as follows:

Table 2: Lean 4 proof modules for trust kernel properties.

Module	Property
`Consensus/BFT`	Safety and liveness under $f < n/3$ Byzantine validators
`Crypto/FROST`	Unforgeability of FROST threshold Schnorr signatures
`Crypto/Threshold/CGGMP21`	Threshold ECDSA composition security and identifiable abort correctness
`Trust/Authority`	Trust delegation chain integrity: no capability escalation, revocation finality

Remark 10.1. *The Lean 4 proofs verify the* protocol logic*, not the implementation. Implementation correctness is established through the type-safe Go and Rust codebases (`luxfi/frost`, `luxfi/threshold`, `luxfi/lattice`) with property-based testing against the Lean 4 reference specifications.*

The proof of trust delegation integrity (Theorem 4.1) is mechanized in `Trust/Authority.lean` as follows: the proof establishes that the authorization graph is acyclic, that revocation propagates to all downstream delegations, and that no delegation chain can exceed the capabilities of its root authority.

11 Related Work

11.1 UCAN (Fission)

User Controlled Authorization Networks (UCANs) [7] implement capability-based authorization using chained JWTs. Each UCAN token contains an issuer, audience, and set of attenuated capabilities. The trust kernel shares the capability delegation model but differs in three ways: (1) capabilities are registered on-chain rather than embedded in bearer tokens, preventing capability replay after revocation; (2) delegation chains are verified by the chain's validator set rather than by the relying party alone; (3) key rotation is enforced by policy rather than requiring token reissuance.

11.2 Ceramic Anchors

Ceramic Network [8] anchors document state on Ethereum, providing a tamper-evident log for mutable documents. The trust kernel generalizes this pattern: where Ceramic anchors document CIDs, the trust kernel anchors *trust decisions* (key lifecycle, governance votes, signatures). This enables audit trails for operations that have no associated document (e.g., a capability revocation or a threshold vote rejection).

11.3 Keybase

Keybase [9] maintained a global key directory with Merkle tree anchoring to Bitcoin. The K-Chain provides equivalent functionality—a global key registry with on-chain anchoring—but extends it with: (1) policy-enforced rotation, (2) ML-KEM wrapping for post-quantum safety, (3) capability-scoped key usage rather than all-or-nothing key trust.

11.4 Signal Protocol Key Management

The Signal Protocol [10] uses the X3DH key agreement and Double Ratchet for forward secrecy. The trust kernel does not replace end-to-end encryption protocols but provides the *root of trust* for key discovery. Where Signal relies on a centralized key server, the trust kernel provides a decentralized, auditable key registry with verifiable rotation history.

12 Conclusion

Lux is not a database. Lux is a trust kernel.

This paper has formalized the trust kernel abstraction: the minimum on-chain substrate that makes sovereign applications composable, auditable, and provider-independent. The four chains—K (key policy), T (threshold governance), M (MPC signing), I (decentralized identity)—provide exactly the trust services that local encrypted vaults cannot provide on their own: global identity, cross-party coordination, and settlement finality.

The trust kernel stores no application data. It stores commitments to decisions about data. Every decision produces a receipt. Receipts are permanent, tamper-evident, and verifiable by any party. This architecture enables capsules to switch providers, delegate capabilities to other capsules, and participate in threshold governance—all without revealing plaintext to the chain or to each other.

The post-quantum migration path ensures that trust decisions made today remain verifiable under future quantum adversaries. The mechanized Lean 4 proofs provide formal guarantees for the critical protocol properties: consensus safety, signature unforgeability, threshold composition security, and trust delegation integrity.

The trust kernel is the answer to a question that the blockchain industry has been asking incorrectly. The question is not "how do we put data on-chain." The question is "what is the

minimum we must put on-chain so that everything else can stay off-chain and still be trusted." Four things: identity, key policy, threshold governance, and receipts.

References

[1] Z. Kelling. Lux Capsule Architecture: Sovereign Encrypted Vaults with Trust Kernel Binding. Lux Industries, 2026.

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

[3] Z. Kelling. Lux ID: Decentralized Identity and Universal Access Management. Lux Industries, 2020.

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

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

[6] Z. Kelling. Lux Warp Messaging: Cross-Chain Communication Protocol. Lux Industries, 2023.

[7] B. Frazee, P. Willison, and B. Holmgren. UCAN: User Controlled Authorization Networks. Fission Codes, 2021.

[8] J. Clark, M. Zargham, and D. Rees. Ceramic: A Decentralized Data Network for Web3. 3Box Labs, 2022.

[9] M. Krohn, M. Mazieres, and C. Coyne. Keybase: Public Key Crypto for Everyone, Publicly Auditable Proofs of Identity. Keybase, Inc., 2014.

[10] T. Perrin and M. Marlinspike. The X3DH Key Agreement Protocol. Signal Foundation, 2016.

[11] C. Komlo and I. Goldberg. FROST: Flexible Round-Optimized Schnorr Threshold Signatures. In *Selected Areas in Cryptography (SAC)*, 2020.

[12] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled. UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts. In *ACM CCS*, 2020.

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

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