LP-Anchor: On-Chain Checkpoint Anchoring for CRDT State Durability

Zach Kelling

Lux Industries

February 2026

Abstract

We describe a mechanism for anchoring CRDT document state to a Lux chain so that clients hold cryptographic proof that a given Merkle root existed at a given logical height. The anchor precompile accepts a 72-byte payload (32-byte app ID, 8-byte height, 32-byte root) and stores it in EVM state at approximately 25,400 gas on C-Chain or near-zero cost on Q-Chain via Quasar DAG parallelism. Users verify anchors by querying the chain and recomputing their local Merkle root. A rollup-style batching mode amortizes cost across 1,000 anchors per on-chain transaction. The design is implemented as a stateful EVM precompile at address `0x0700...10` in the Lux precompile registry (Privacy/Encryption page, item 0x10).

Contents

1 Problem Statement

Base (and any CRDT-based application) replicates state across clients via conflict-free merge operations. The server is the coordination point: it receives operations, merges them, and fans out updates. This creates a trust asymmetry:

1. The server can **withhold operations**: silently drop writes from a subset of clients.
2. The server can **rewrite history**: present different state vectors to different clients without detection.
3. The server can **lie about time**: claim a state snapshot existed at time T when it did not.

The user needs a *tamper-evident durability backstop*: cryptographic proof that the CRDT state at logical height h had Merkle root r, anchored to a public chain whose finality is independent of the Base server.

Definition 1.1 (Anchor)**.** *An anchor is a tuple* $(appID, h, r, t, \sigma)$ *where* $appID \in \{0,1\}^{256}$ *identifies the application,* $h \in \mathbb{N}$ *is the logical height (monotonically increasing),* $r \in \{0,1\}^{256}$ *is the Merkle root of the CRDT state at height* h*,* t *is the chain timestamp at inclusion, and* σ *is the submitter's ECDSA signature (implicit in the EVM transaction).*

2 Architecture

2.1 Anchor Lifecycle

1. **Compute**: Application computes SHA-256 Merkle root of serialized CRDT `DocumentSnapshot` every N operations or T seconds (configurable; default $N = 1000$, $T = 300$s).
2. **Submit**: Application calls `anchor.Submit(appID, height, root)` which encodes a 72-byte payload and sends an EVM transaction to the anchor precompile.
3. **Finalize**: Transaction is included in a block and finalized by Quasar certificate (Q-Chain) or Snow consensus (C-Chain). The anchor is now immutable.
4. **Verify**: Any party calls `anchor.Get(appID, height)` via `eth_call`, recomputes the local Merkle root from their CRDT state, and compares.

2.2 Precompile Design

The anchor precompile is a stateful EVM precompile registered at address `0x0700...10` (Privacy/Encryption page, C-Chain slot, item 0x10 in the LP addressing scheme). It exposes four functions via ABI selector:

Function	Selector	Gas
`submit(bytes32,uint64,bytes32)`	`0x3c21e8a1`	25,400
`get(bytes32,uint64)`	`0x8eaa6ac0`	2,600
`getLatest(bytes32)`	`0xba41d847`	2,600
`list(bytes32,uint64,uint64)`	`0x70fa9a4d`	2,600 + 200/entry

2.2.1 Storage Layout

Each anchor occupies two EVM storage slots:

- Slot `keccak256`$(appID \| h)$: the 32-byte root.
- Slot `keccak256`$(appID \|$ `"latest"`$)$: the highest submitted height for this appID (8 bytes, right-padded).

This gives $O(1)$ lookup per anchor and $O(1)$ latest-height query. The `list` function iterates heights $[h_{\text{from}}, h_{\text{to}})$, issuing one `SLOAD` per height.

2.2.2 Gas Cost Derivation

$$\begin{aligned} \text{submit gas} &= G_{\text{base}} + 2 \times G_{\text{sstore\_new}} + G_{\text{keccak}} \\ &= 400 + 2 \times 12{,}000 + 600 + 400 \\ &= 25{,}400 \end{aligned}$$

where $G_{\text{sstore\_new}} = 12{,}000$ for writing a new storage slot (after EIP-2929 warm access), $G_{\text{keccak}} = 600$ for the two hash computations, and $G_{\text{base}} = 400$ for ABI decoding and validation. A `get` call costs 2,600 (one warm `SLOAD`).

2.3 Height Monotonicity Invariant

The precompile enforces $h > h_{\text{latest}}$ for a given appID. Attempts to submit a height $\leq$ the current latest revert with `AnchorHeightNotMonotonic`. This prevents retroactive anchor forgery: once height 100 is anchored, no one can insert a different root at height 100 or below.

3 Why Q-Chain

The Q-Chain (Quantum Chain) is the preferred anchor target for three reasons.

PQ-safe finality. Q-Chain blocks carry Quasar certificates: triple-signed (BLS + ML-DSA + Pulsar) finality proofs. An anchor finalized by a QuasarCert inherits post-quantum security. C-Chain finality relies on Snow consensus (probabilistic, classical signatures only).

DAG parallelism. Q-Chain processes transactions as a DAG, not a sequential block. Anchor submissions from independent appIDs are conflict-free and process in parallel. On C-Chain, all transactions compete for sequential block space.

Cost. Q-Chain transaction fees are validator-set-determined and currently near zero for data-only transactions (no EVM execution). A C-Chain anchor at 25,400 gas $\times$ 25 nLUX/gas = 635 nLUX ($\approx$ \$0.001 at \$1.50/LUX). Q-Chain is 10–100$\times$ cheaper for equivalent data anchoring because it avoids EVM metering overhead.

C-Chain alternative. If the application already pays C-Chain gas (e.g., a DeFi protocol on C-Chain), using the same chain for anchoring avoids cross-chain complexity. The precompile is registered on both chains.

4 Anchor Transaction Format

The on-chain payload is 72 bytes:

Field	Bytes	Encoding
appID	32	`bytes32` (SHA-256 of app name)
height	8	`uint64`, big-endian
root	32	`bytes32` (Merkle root)

Total: 72 bytes of calldata + 4-byte function selector = 76 bytes. The submitter identity is `msg.sender` (implicit ECDSA signature in the EVM transaction envelope). No additional signature field is needed.

5 Verification Path

A user who suspects the server is lying:

1. Queries `getLatest(appID)` to learn the highest anchored height $h_{\max}$.
2. Queries `get(appID, h)` for the height of interest, yielding root r_{chain}.
3. Serializes their local CRDT `DocumentSnapshot` at logical height h and computes r_{local} = SHA-256(snapshot).
4. Compares: $r_{\text{local}} = r_{\text{chain}}$ implies state integrity at height h.
5. If $r_{\text{local}} \neq r_{\text{chain}}$, the server diverged from the anchored state. The user holds cryptographic evidence of tampering.

Verification is a read-only `eth_call`—zero gas, zero latency beyond RPC round-trip (< 50ms to a local node, < 200ms cross-region).

6 Adversary Model

Theorem 6.1 (Anchor Integrity)**.** *Let $\mathcal{A}$ be an adversary controlling the Base server but not $\geq 1/3$ of chain validators. $\mathcal{A}$ cannot produce an anchor $(appID, h, r')$ for $r' \neq r$ at height h if an honest anchor $(appID, h, r)$ was already finalized.*

Proof. The height monotonicity invariant prevents overwriting: once height h is anchored, no transaction with height $\leq h$ for the same appID is accepted. Reverting the anchor requires a chain reorganization past the Quasar finality depth, which requires compromising $\geq 1/3$ of validators across all three signature schemes (BLS, ML-DSA, Pulsar). □

Withholding attack. The server can refuse to anchor, or anchor selectively. This is detectable: if the user expects an anchor at height h and `getLatest(appID)` returns $h' < h$, the server is withholding. Mitigation: let clients submit anchors directly (the precompile accepts any `msg.sender`).

Equivocation. The server anchors root r_1 at height h on-chain but presents root r_2 to a client. The client detects this on verification (step 4 above). The on-chain record is authoritative.

Reorg protection. On Q-Chain, a QuasarCert-finalized anchor is irreversible under the Quasar security model. On C-Chain, wait for 2 block confirmations ($\approx$ 4s) for probabilistic finality equivalent to 10^{-9} reorg probability.

7 Cost Analysis

Chain	Gas/anchor	Cost (LUX)	Cost (USD)
C-Chain	25,400	635 nLUX	\$0.001
Q-Chain	≈ 0	< 1 nLUX	<\$0.000001

Frequency trade-off. Anchoring every 5 minutes = 288 anchors/day = \$0.29/day on C-Chain, essentially free on Q-Chain. Anchoring every 1,000 operations at 100 ops/sec = anchor every 10 seconds = 8,640/day = \$8.64/day on C-Chain.

Recommendation. Default to every 5 minutes *and* every 1,000 operations (whichever comes first). For cost-sensitive deployments, use Q-Chain. For EVM-native apps already on C-Chain, use C-Chain to avoid cross-chain queries.

8 Batched Anchoring

For applications managing many documents (e.g., a multi-tenant Base deployment), individual anchors per document are wasteful. Instead:

1. Collect up to 1,000 per-document Merkle roots.
2. Build a Verkle tree (or SHA-256 Merkle tree) over these roots.
3. Submit one anchor: $(appID_{\text{batch}}, h_{\text{batch}}, r_{\text{batch}})$.
4. Publish the Verkle proof for each document's root as an off-chain inclusion witness.

This amortizes the 25,400 gas cost across 1,000 documents: 25.4 gas per document. The off-chain Verkle proof is ≈ 1 KB and can be stored alongside the CRDT state.

Batching is a client-side optimization. The precompile does not change; it anchors one root per call regardless of whether that root represents one document or a thousand.

9 Implementation

9.1 Precompile

The precompile is implemented as a Go module at `github.com/luxfi/precompile/anchor`, registered in the Lux precompile registry. It implements `contract.StatefulPrecompiledContract` with four ABI-dispatched functions.

Storage keys use `keccak256(appID || height)` for root storage and `keccak256(appID || 0xFF..FF)` for the latest-height slot. The sentinel `0xFF..FF` avoids collision with any valid height (uint64 max is $2^{64} - 1$, which fits in 8 bytes and does not collide with the 32-byte sentinel).

9.2 Client SDK

The client SDK at `github.com/hanzoai/base/crdt` provides:

```
type Anchorer interface {
    Submit(ctx context.Context, root [32]byte) error
    Verify(ctx context.Context, height uint64, root [32]byte) (bool,
        error)
}
```

An `RPCAnchorer` implementation talks to the chain via JSON-RPC. A background goroutine anchors on a configurable interval.

10 Comparison with Prior Art

	LP-Anchor	**Ceramic**	**KERI**	**Radicle**	**Nostr**
PQ-safe finality	Yes (Q-Chain)	No	No	No	No
On-chain storage	2 slots/anchor	Tx log	Off-chain	Tx log	None
Batching	Verkle tree	Merkle tree	KEL	Git tree	N/A
Verify latency	< 200ms	Seconds	Seconds	Seconds	N/A
Gas/anchor	25.4k	$\sim$50k	N/A	$\sim$30k	N/A

Ceramic anchors CIDs to Ethereum via a Merkle tree of stream tips, batching every 30 minutes. KERI maintains key event receipt logs off-chain with witness infrastructure. Radicle and Gitopia anchor Git commit hashes. Nostr event IDs are self-certifying but have no chain finality.

LP-Anchor is the only design with post-quantum finality guarantees and sub-second verification latency.

11 Conclusion

LP-Anchor provides a minimal, high-assurance mechanism for anchoring CRDT state to a public blockchain. The design requires 72 bytes per anchor, 25,400 gas on C-Chain (near-zero on Q-Chain), and delivers sub-200ms verification latency. The height monotonicity invariant and Quasar finality guarantee that anchored state cannot be retroactively altered without compromising the chain's validator set across three independent cryptographic assumptions.

The precompile is general: any application that computes a periodic state hash can use it. Base CRDT anchoring and `liquidity/state` reconciliation anchoring are the first two consumers.

References

[1] Z. Kelling, "LP-020: Quasar — Post-Quantum BFT Consensus," *Lux Research*, April 2026.

[2] M. Shapiro et al., "Conflict-free Replicated Data Types," *Proc. SSS*, 2011.

[3] Ceramic Network, "Ceramic Protocol Specification v2," https://ceramic.network, 2024.

[4] S. Smith, "Key Event Receipt Infrastructure (KERI)," *IETF Draft*, 2022.

[5] V. Buterin, M. Swende, "EIP-2929: Gas cost increases for state access opcodes," *Ethereum Improvement Proposals*, 2021.