NilStore Network: A Protocol for Decentralized, Verifiable, and Economically Efficient Storage

Contents

NilStore Network: A Protocol for Decentralized, Verifiable, and Economically Efficient Storage

(White Paper v1.0)

Date: 2025-10-02 **Authors:** NilStore Core Team

Abstract

NilStore is a decentralized storage network designed to provide high-throughput, verifiable data storage with significantly reduced operational overhead. It leverages a novel consensus mechanism based on Proof-of-Useful-Data (PoUD) and Proof-of-Delayed-Encode (PoDE), utilizing plaintext storage verified via KZG commitments and timed derivations (Argon2id). By employing a topological data placement strategy (Nil-Lattice), NilStore drastically lowers the hardware barrier to entry, enabling participation from edge devices to data centers. This paper details the system architecture, the NilFS data abstraction layer, the Nil-Mesh routing protocol, the $STOR-Only economic model, the hybrid L1/L2 settlement architecture, and the underlying cryptographic primitives, including the Nil-VRF epoch beacon, that secure the network.

1. Introduction

1.1 Motivation

While existing decentralized storage protocols have demonstrated the viability of incentive-driven storage, they often rely on computationally intensive Proof-of-Replication (PoRep) stacks requiring significant GPU investment. This centralizes the network around large-scale operators and increases the total cost per byte.

NilStore retains strong cryptographic guarantees while eliminating the "sealing" process, reducing the time to onboard storage to minutes on standard CPUs. This democratization of access increases network resilience through geographic distribution and enables a more efficient storage marketplace.

1.2 Key Innovations

- **Plaintext possession as first-class:** Storage providers (SPs) keep the **cleartext** bytes of assigned Data Units (DUs) on disk and prove possession regularly with near-certain full-coverage over time.
- **PoUD + PoDE: PoUD** (KZG-based Provable Data Possession over DU cleartext) + **PoDE** (timed window derivations using Argon2id) are the **normative** per-epoch proofs.
- **Nil-Mesh Routing:** Heisenberg-lifted K-shortest paths for optimized latency and Sybil resistance.
- **$STOR-Only Economy:** A unified economic model using $STOR for capacity commitment, bandwidth settlement, and governance, explicitly excluding external price oracles.
- **Hybrid Settlement:** A specialized L1 (DA Chain) for efficient proof verification bridged via ZK-Rollup to an EVM L2 for liquidity and composability.
- **Nil-VRF Epoch Beacon:** A BLS12-381-based Verifiable Random Function (VRF) with BATMAN threshold aggregation to ensure unbiased, unpredictable epoch challenges.

2. System Architecture

NilStore employs a hybrid architecture that decouples Data Availability (DA) consensus from economic settlement, optimizing for cryptographic efficiency and ecosystem composability.

2.1 Architectural Layers

1. **Data Layer (NilFS):** Handles object ingestion, erasure coding, and placement.
2. **Network Layer (Nil-Mesh):** Manages peer discovery, routing, and QoS measurement.
3. **Consensus Layer (DA Chain - L1):** Verifies **PoUD (KZG multi-open) + PoDE timing attestations**, manages stake, mints rewards, and generates the epoch beacon (Nil-VRF).
4. **Settlement Layer (L2 Rollup):** Handles economic transactions, liquidity, and governance.

2.2 The DA Chain (L1)

The Data Availability Chain is a minimal L1 (built using Cosmos-SDK/Tendermint BFT) optimized for NilStore's cryptographic operations.

- **Function:** Verifying **KZG openings (multi-open)** for PoUD, enforcing PoDE timing bounds via watcher digests, managing $STOR staking, executing slashing logic, and running the Nil-VRF beacon. It does not run a general-purpose VM.
- **Required pre-compiles (normative):** (a) **BLAKE2s-256**, (b) **Poseidon** (for Merkle paths), (c) **KZG** (G1/G2 ops; multi-open on BLS12-381), (d) **VDF Verification**, and (e) **BLS Signature Verification/Pairing** (for Nil-VRF).
- **Rationale:** The intensive cryptographic operations required for daily proof verification and beacon generation are best handled natively.

2.3 The Settlement Layer (L2)

Settlement occurs on a ZK-Rollup (using PlonK/Kimchi) bridged to a major EVM ecosystem.

- **Function (One-Token Profile):** Manages $STORonly**, mintsDealNFTs, hoststheNilDAO, andea **STOR-denominated** settlement for storage and bandwidth. **Non-$STOR assets are out-of-scope** for protocol contracts; any conversions happen off-protocol.

2.4 The ZK-Bridge

The L1 aggregates epoch verification results into a single proof/digest and posts it to the L2 bridge contract. **Normative circuit boundary**:

1. **Public inputs**: `{epoch_id, DA_state_root, poud_root, bw_root, validator_set_hash}`.
2. **Verification key**: `vk_hash = sha256(vk_bytes)` pinned in the L2 bridge at deployment. Upgrades require DAO action and timelock. An Emergency Circuit MAY perform expedited **VK-only** upgrades under specific conditions (See Section 9.2).
3. **State mapping**: On accept, the bridge **atomically** updates `{poud_root, bw_root, epoch_id}`; monotonic `epoch_id` prevents replay.

4. **Failure domains**: Mismatches or non-monotonic epochs initiate a Grace Period (default 24h). If unresolved, the bridge halts. Validity is enforced by the proof and pinned `vk_hash`, requiring no trusted relayers.
5. **Proof Generation (Normative):** The ZK proof MUST be generated by a decentralized prover network or a rotating committee selected from the L1 validator set, with slashing penalties for failure to submit valid proofs.

3. Data Layer (NilFS)

NilFS abstracts data management complexity, automating the preparation, distribution, and maintenance of data. SPs and Users interact with Data Units (DUs), not raw files or replication strategies.

3.1 Object Ingestion and Data Units (DUs)

1. **Content-Defined Chunking (CDC):** Ingested objects are split using CDC (e.g., Rabin fingerprinting) to maximize deduplication. Chunks are organized into a Merkle DAG (CIDv1 compatible).
2. **Data Unit Packing:** Chunks are serialized and packed into standardized **Data Units (DUs)**. DU sizes are powers-of-two (1 MiB to 8 GiB).
3. **Commitment:** The client computes a DU KZG commitment (`C_root`) over the serialized data (See Section 6.2.1). This `C_root` binds the content for all future proofs.

3.2 Erasure Coding and Placement

Normative (Redundancy & Sharding). Each DU is encoded with **systematic Reed–Solomon over GF(2^8)** using **symbol_size = 1 KiB**. The default profile is **RS(n=12, k=9)** (**1.33×** overhead), tolerating **f = 3** arbitrary shard losses.

3.2.1 Deterministic Placement (Nil-Lattice) Shards are placed on a directed hex-ring lattice (Nil-Lattice) to maximize topological distance and network resilience.

- **Coordinate Calculation:** The coordinate (r,) for shard `j` is determined by: `pos := Hash(CID_DU  ClientSalt_32B  j) → (r, )`
- **Anti-Grinding (Normative):** `ClientSalt_32B` MUST be derived deterministically from the client's signature over the Deal parameters (e.g., `Blake2s-256("NILSTORE-SALT-V1" || Sig_Client)`) to prevent placement grinding.
- **Placement constraints (Normative):** At most **one shard per SP per ring-cell**. Shards of the same DU MUST be placed across distinct ring-cells with a minimum topological separation (governance-tunable).

3.2.2 Advanced Profiles (Optional)

- **RS-2D-Hex Profile:** Couples two-dimensional erasure coding with the Nil-Lattice. It maps row redundancy to radial rings and column redundancy to angular slices, enabling O(|DU|/n) repair bandwidth under churn.

- **Durability Dial Abstraction:** Exposes a user-visible `durability_target` (e.g., 11 nines) that deterministically resolves to a governance-approved redundancy profile (RS-Standard or RS-2D-Hex).

3.3 Autonomous Repair Protocol

The network autonomously maintains durability through a commit-reveal bounty system.

1. **Detection:** If DU availability drops below the resilience threshold (e.g., k+1), a `RepairNeeded` event is triggered.
2. **Commit-Reveal:** Repair nodes reconstruct the missing shards and submit solutions in a two-phase process.
3. **Verification and Bounty:** The L1 chain verifies the solution. The Resilience Bounty (default: 5% of the remaining escrowed fee) is awarded to the earliest valid commitment.
   - **Normative (Commitment Drift Prevention):** The repair solution MUST include openings against the original DU KZG commitment (`C_root`). Re-committing a DU is invalid.
4. **Anti-withholding (Normative):** The SP originally assigned the failed shard incurs an immediate penalty strictly greater than the repair bounty (Default: 1.5 × Bounty) and an automatic demerit. The penalty for triggering a repair is significant (Default: 25% of the DU collateral).
5. **Collocation Filter (Normative):** To prevent SPs from repairing their own dropped shards, a filter disqualifies identities that were recently assigned the shard, are within the same IP subnet (/24 IPv4 or /48 IPv6), the same ASN, or exhibit statistically significant RTT Profile Similarity (See Section 4.2).

4. Network Layer (Nil-Mesh)

Nil-Mesh is the network overlay optimized for low-latency, topologically aware routing, utilizing the geometry of the Nil-Lattice.

4.1 Heisenberg-Lifted Routing

- **Mechanism:** Peer IDs (NodeIDs) are mapped ("lifted") to elements in a 2-step nilpotent Lie group (Heisenberg-like structure) corresponding to their lattice coordinates.
- **Pathfinding:** K-shortest paths (K=3) are computed in this covering space and projected back to the physical network. This offers superior latency compared to standard DHTs.
- **Sybil Resistance:** This approach increases Sybil resistance by requiring attackers to control entire topological regions ("Ring Cells").

4.1.1 Secure Identity Binding (Normative) Peer IDs are securely bound to lattice coordinates (r,) through a costly registration process, preventing rapid movement and ensuring that capturing a Ring Cell requires significant capital and time. To register or move, an SP MUST:

1. Bond a minimum amount of $STOR (Stake_Min_Cell), specific to the target Ring Cell.
2. Compute a Verifiable Delay Function (VDF) proof: `Proof_Bind = VDF(NodeID, r, , difficulty)`.

The DAO MUST periodically update `Stake_Min_Cell` and VDF `difficulty`, raising them automatically if empirical concentration increases.

4.2 RTT Attestation and QoS Oracle

Verifiable Quality of Service (QoS) is crucial for performance, path selection, and preventing Sybil self-dealing (verifying RTT > network floor).

- **Attestation:** Nodes continuously monitor and sign Round-Trip Time (RTT) attestations with peers.
- **On-Chain Oracle:** A **stake-weighted attester set** posts RTT digests (Poseidon Merkle roots) to the DA chain.

Normative Oracle Procedures:

1. **Challenge-response**: Clients/attesters issue random tokens; SPs must echo tokens within `T_max`.
2. **VDF Enforcement (Mandatory):** Every attestation MUST include a short-delay VDF proof. The VDF input MUST include the random challenge token, computed after receiving the challenge and before transmitting the response, proving the delay occurred within the RTT measurement window and preventing pre-computation.
3. **Diversity & rotation**: The attester set MUST achieve a minimum diversity score (ASN/Region distribution) and assignments are epoch-randomized and committed on-chain.
4. **Slashing**: Equivocation or forged attestations are slashable via on-chain fraud proofs.
5. **Sybil control**: Weight attesters using **quadratic weighting** (weight $\sqrt{\text{STOR}}$). The total weight of any single entity or correlated group (defined by ASN/Region cluster or RTT Profile Similarity) MUST NOT exceed 20% of the total attester weight.

5. Economic Model ($STOR-Only)

NilStore employs a unified token economy ($STOR) to align long-term security incentives with network utility. The protocol explicitly avoids in-protocol stablecoins and external price oracles.

5.1 $STOR (Staking and Capacity Token)

- **Supply:** Fixed (1 Billion).
- **Functions:** Staking collateral for SPs and Validators; medium of exchange for storage capacity and bandwidth; governance voting power.
- **Sink:** Slashing events.

5.2 Fee Market for Bandwidth ($STOR-1559)

One-token profile (normative): The protocol uses **$STOR only** for bandwidth settlement. **No activity-based inflation** is permitted.

Each region `r` and epoch `t` defines **BaseFee_r,t** (in $STOR per MiB), adjusted EIP-1559-style toward a byte-throughput target U*. For a payable origin→edge transfer of `b` bytes:

```
Burn      = BaseFee[r,t] × b        // burn in $STOR
```

`Payout = PremiumPerByte × b // pay provider in $STOR`

Update rule (bounded): `BaseFee_{t+1} = BaseFee_t · (1 + · (U_t − U*) / U*)` with bounds (default ±12.5%).

Protocol currency invariant: Settlement and escrow contracts **MUST accept $STOR only**.

5.3 Off-Protocol Payer Adapters

Wallets, edges, and merchant gateways MAY implement **off-protocol adapters** that quote human-readable prices off-chain, acquire $STOR via external venues, and fund payer * *STOR escrow**. These adapters are not part of consensus.

6. Consensus and Verification

The economic model is enforced cryptographically through the PoUD+PoDE consensus mechanism on the L1 DA Chain, driven by the Nil-VRF epoch beacon.

6.1 Overview of the Consensus Model

NilStore utilizes a **Plaintext-only** proof mode. The objective is to attest, per epoch, that an SP (a) stores the **cleartext** bytes of their assigned DU intervals (PoUD) and (b) can perform **timed, beacon-salted derivations** over randomly selected windows quickly enough that fetching from elsewhere is infeasible within the proof window (PoDE).

Security anchors: (i) DU **KZG commitment** `C_root` recorded at deal creation. (ii) BLS-VRF epoch beacon for unbiased challenges (Section 6.4). (iii) On-chain **KZG multi-open** pre-compiles. (iv) Watcher-enforced timing digests.

6.2 Proof-of-Useful-Data (PoUD)

PoUD verifies the content correctness of the stored data against the original commitment.

6.2.1 DU Representation & Commitment (Normative) A DU is segmented logically into **1 KiB symbols**. To commit the data using KZG (which operates over the BLS12-381 scalar field), the DU data MUST be serialized and chunked into 31-byte elements. Each chunk is interpreted as an integer (little-endian) and embedded as a field element. The KZG commitment `C_root` is computed over the polynomial formed by these field elements.

All KZG operations MUST utilize a common, pinned Structured Reference String (SRS) generated via a verifiable Multi-Party Computation (MPC) ceremony (e.g., Perpetual Powers of Tau).

6.2.2 PoUD Mechanism For each epoch and each assigned DU sliver interval:

1. **Challenge Derivation:** The protocol expands the epoch beacon (Section 6.4) to select `q` distinct, unpredictable symbol indices.
2. **Proof Submission:** The SP MUST provide one or more **KZG multi-open** proofs at the chosen 1 KiB symbol indices proving membership in the DU commitment `C_root`. SPs SHOULD batch using multi-open to minimize calldata.
3. **Verification:** L1 verifies KZG openings using the precompile (Section 2.2).

6.3 Proof-of-Delayed-Encode (PoDE)

PoDE enforces timed locality, ensuring the SP is actively processing the data and not retrieving it on demand. It uses Argon2id for its memory hardness and sequentiality.

6.3.1 The `Derive` Function (Normative) PoDE relies on the `Derive` function to deterministically compress a cleartext DU window into a verifier-recomputable digest, domain-separated by the epoch beacon.

Algorithm (Argon2id Sequential):

```
Derive(clear_window: bytes, beacon_salt: bytes, row_id: u32, epoch_id: u64, du_id: u128) -

tag  = "PODE_DERIVE_ARGON_V1"
salt = Blake2s-256(tag ‖ beacon_salt ‖ u32_le(row_id) ‖ u64_le(epoch_id) ‖ u128_le(du_id))
// Hash input to ensure high entropy input to Argon2id
input_digest = Blake2s-256("PODE_INPUT_DIGEST_V1" ‖ clear_window)
// Parameters (H_t, H_m, H_p) defined by Dial Profile (See Appendix A).
// H_p MUST be strictly 1 to enforce sequentiality.
leaf64 = Argon2id(password=input_digest, salt=salt, t_cost=H_t, m_cost=H_m, parallelism=1,
return leaf64
```

PoDE Recalibration (Normative): The NilDAO MUST periodically recalibrate `H_t` and `H_m` parameters to maintain the target `Δ_work` (1s) based on baseline hardware performance.

6.3.2 PoDE Mechanism

1. **Challenge Derivation:** The protocol selects `R` PoDE windows of size `W = 8 MiB` (governance-tunable).
2. **Timed Derivation:** The SP MUST compute `deriv = Derive(clear_bytes[interval], beacon_salt, row_id, ...)` within the proof window. The proof includes `H(deriv)` and the clear bytes needed for recomputation.
3. **PoDE Linkage (Normative):** The prover MUST include a KZG opening proof `_kzg` demonstrating that the `clear_window` input bytes correspond exactly to the data committed in `C_root`.
4. **Concurrency & volume (Normative):** The prover MUST satisfy at least `R` parallel PoDE sub-challenges per proof window (default `R  16`). The aggregate verified bytes per window MUST be `B_min` (default `B_min  128 MiB`).
5. **Verification:** L1 verifies `_kzg`. Watchers enforce timing via RTT-Oracle and publish pass/fail digests.

6.4 Nil-VRF and Epoch Beacon

NilStore requires unbiased, unpredictable randomness for selecting PoUD/PoDE challenges and the retrieval sampling set. This is provided by the Nil-VRF, a BLS12-381-based Verifiable Random Function (VRF).

6.4.1 Design Choice The Nil-VRF is instantiated as a **BLS-signature-based VRF**: VRF proofs are BLS signatures on `hash_to_G2(msg)`, and verification is a single pairing check. This

design is:

- **Uniquely provable:** A single, deterministic proof per `(pk,msg)`.
- **Deterministically verifiable** on-chain efficiently.
- **Aggregate-friendly:** Supports BATMAN threshold aggregation (2/3 honest).

We follow RFC 9380 for `hash_to_G2` with a NilStore-specific DST: `"BLS12381G2_XMD:SHA-256_SSWU_RO_NIL_`

6.4.2 BATMAN Threshold Aggregation (t 2/3) To ensure beacon liveness and security, NilStore uses BATMAN aggregation.

- **Setup:** Committee size `N`; threshold `t = 2N/3` . Master key is split using polynomial secret sharing. Participants MUST provide Proof of Possession (PoP) during registration to prevent rogue-key attacks.
- **Per-epoch share posting:** Each participant `i` publishes their share `(pk_i, _i)` where `_i = sk_i · H(epoch_ctr)`.
- **Aggregation:** The aggregator collects `t` valid shares and computes the aggregate proof using Lagrange interpolation: `_agg = Σ _i · _i`.

6.4.3 Deterministic Share-Selection (Normative) To prevent the aggregator from grinding the beacon by subset selection:

1. Participants post shares on L1 before the deadline `_close`.
2. The aggregator identifies the candidate set.
3. **Grinding Mitigation:** Let `Seed_select` be the finalized beacon of the previous epoch (`beacon_{t-1}`).
4. **Canonical Set Definition:** Compute `Score_i = HMAC-SHA256(Key=Seed_select, Message=share_id_i)`. The canonical aggregation set is strictly the `t` shares with the lowest `Score_i` values.
5. The aggregator MUST use this canonical set to compute `( _agg, pk_agg)`.

6.4.4 On-chain Verification & Beacon Derivation The L1 chain verifies the aggregate proof using a single pairing check.

```
require( pairing(pkAgg, H) == pairing(G1_GEN, piAgg) );
y  = blake2s256("NIL_VRF_OUT" ‖ compress(pkAgg) ‖ compress(H) ‖ compress(piAgg));
beacon_t = blake2s256("NIL_BEACON" ‖ y);
```

The 32-byte `beacon_t` feeds PoUD/PoDE challenge derivation (Section 6.2.2, 6.3.2) and seeds the retrieval-sampling RNG (Section 6.6.1).

6.5 Proof Coverage and Parameters

PDP-PLUS Coverage SLO (normative). Define CoverageTargetDays (default 365). The governance scheduler MUST choose per-epoch index sets (challenge rate q/M) so that for every active DU, the expected fraction of uncovered bytes $(1 - q/M)^T$ after T=CoverageTargetDays is 2^{-18}.

The scheduler MUST be commit-then-sample: indices for epoch `t` are pseudorandomly derived from the epoch beacon and are not known to SPs before the deadline of epoch `t-1`.

6.6 Retrieval Receipts and QoS Auditing

To account for bandwidth and ensure Quality of Service, clients sign receipts upon successful retrieval, which are then probabilistically sampled for verification.

- **Receipt Schema (Normative):** `Receipt := { CID_DU, Bytes, EpochID, ChallengeNonce, ExpiresAt, Tip_BW, Miner_ID, Client_Pubkey, Sig_Ed25519 [, GatewaySig?] }`
- **Verification model:** Ed25519 signatures are verified off-chain by watchers and/or on the DA chain. The protocol commits to a **Poseidon Merkle root** of receipts (`BW_root`) and proves byte-sum consistency.
- **Commit requirement (Normative):** For epoch `t`, SPs MUST have posted `BW_commit := Blake2s-256(BW_root)` by the last block of epoch `t-1`. Failure to post forfeits all bandwidth payouts for `t`.

6.6.1 Probabilistic Retrieval Sampling A governance-tunable fraction of receipts (default 5%) are verified each epoch.

1. **Sampling Seed (Normative):** Derived from the epoch beacon: `seed_t := Blake2s-256("NilStore-Sample" ‖ beacon_t ‖ epoch_id)`
2. **Abuse Score Calculation:** Calculate an Abuse Score `A_score(SP)` based on historical failures, RTT anomalies, volume spikes, and RTT Profile Similarity.
3. **Risk-Based Sampling (Normative):** The global sampling fraction `p` is tunable, but the per-SP sampling rate `p_sp` MUST be dynamically adjusted based on `A_score(SP)`.
4. **Honeypot DUs:** DUs created and funded pseudonymously (via a blinded pool) to mimic organic traffic. Any retrieval receipt for a Honeypot DU is automatically selected for 100% verification.

6.6.2 Verification and Enforcement Watchers verify signatures, nonces, RTT transcripts (via QoS Oracle), and inclusion in `BW_root`.

- **Fail (Minor):** If failures (default 1%), deduct failing receipts and **forfeit all retrieval payouts** for the epoch.
- **Fail (Major):** If failures > , forfeit payouts and apply quadratic slashing to bonded $STOR.

6.6.3 Verification Load Cap and Economic Circuit Breaker (Normative) The total on-chain verification load MUST be capped (DAO-tunable) to prevent DoS. If the cap is reached during a security escalation:

1. **Prioritize High-Risk Receipts:** Sampling prioritizes receipts associated with high abuse scores.
2. **Source Verification Costs:** Excess costs are sourced from the Security Treasury.
3. **Dynamic BaseFee Adjustment:** If the Treasury is insufficient, the protocol MUST dynamically increase the `BaseFee` (Burn component) (Section 5.2) for the duration of the escalation. Payouts MUST NOT be throttled.

7. The Deal Lifecycle

7.1 Quoting and Negotiation (Off-Chain)

Clients query Nil-Mesh for SPs near the required lattice slots. SPs respond with Quotes including price, collateral requirements, and QoS caps. The client selects the optimal bundle using the QoS Oracle.

7.2 Deal Initiation (On-Chain - L2)

1. **`CreateDeal`:** Client calls the function on the L2 settlement contract, posts the Commitment Root (C_root), locks the total storage fee in $STOR escrow, and mints a **Deal NFT** (ERC-721).
2. **`MinerUptake`:** The selected SP bonds the required $STOR collateral and commences service.
3. **`StorageAttest`:** The SP MUST post an attestation tuple `{sector_id, origin_root, deal_id}` committing to the data layout before proofs are counted toward vesting.

7.3 Vesting and Slashing

- **Vesting:** The escrowed fee is released linearly to the SP each epoch, contingent on a valid **PoUD + PoDE** submission.
- **Consensus Parameters (Normative):**
  - **Epoch Length (`T_epoch`)**: 86,400 s (24 h).
  - **Proof Window (`Δ_submit`)**: 30 s after epoch end (network scheduling window).
  - **Per-replica Work Bound (`Δ_work`)**: 1 s (baseline PoDE calibration).

7.3.1 Slashing Rule (Normative) Missed **PoUD + PoDE** proofs trigger a quadratic penalty on the bonded $STOR collateral, augmented by a correlation factor to penalize clustered failures.

```
Penalty = min(0.50, 0.05 × (Consecutive_Missed_Epochs)²) × Correlation_Factor(F)
```

- **Correlation_Factor(F):**
  - Let $F_{cluster}$ be the fraction of total capacity within a diversity cluster (ASN×region cell, or collocated identities) that failed.
  - $Correlation\_Factor(F) = 1 + \alpha \cdot (F_{cluster})^{\beta}$ (where $\beta \geq 2$ for superlinear penalty).
  - The Correlation_Factor is capped (e.g., 5x).

7.4 Advanced Mechanisms

- **Bandwidth-Driven Redundancy (Normative):** NilStore aligns replica count with observed demand (Heat Index). When demand crosses a threshold, a VRF committee assigns short-TTL "hot replicas" to additional providers chosen by weighted rendezvous hashing based on Provider Capability Vectors (PCV).
- **Multi-Stage Epoch Reconfiguration:** Ensures uninterrupted availability during committee churn by directing writes to epoch e+1 immediately, while reads remain served by epoch e until the new committee reaches a readiness quorum (signaled on L1 and verified by watchers).

8. Advanced Features: Spectral Risk Oracle ( )

To manage systemic risk, NilStore incorporates an on-chain volatility oracle ( ).

- **Mechanism:** is calculated daily from the Laplacian eigen-drift of the storage demand graph (tracking object-to-region flows), filtered to exclude manipulative patterns (Sybil filtering, high abuse scores). `_t := ||Δ ..k(Graph_t)||`
- **Application (Dynamic Collateral):** The required collateral for a deal is dynamically adjusted based on internal network volatility ( ). External price volatility is explicitly excluded. `Required_Collateral := Base_Collateral · f( )`
- **Oracle Management (Normative):** $f(\sigma)$ MUST incorporate dampening (e.g., 30-day EMA). The rate of change in Required_Collateral MUST be capped per epoch (e.g., max 10% increase). A grace period (default 72h) is provided for collateral top-ups before liquidation.

9. Governance (NilDAO)

The network is governed by the NilDAO, utilizing stake-weighted ($STOR) voting on the L2 Settlement Layer.

9.1 Scope

The DAO controls:

- **Economic parameters:** Slashing ratios, bounty percentages, BaseFee adjustment bounds.
- **QoS sampling dials:** `p` (sampling fraction), (tolerance), `_sys` (system anomaly rate).
- **PoDE/PoUD pressure dials:** `R` (parallel sub-challenges), `B_min` (minimum verified bytes), and PoDE calibration (`H_t`, `H_m`).
- **Network parameters:** Durability Dial mapping, reconfiguration thresholds, Verification Load Cap.
- **Network upgrades and the treasury.**

9.2 Upgrade Process

- **Standard Upgrades:** Require a proposal, a voting period, and a mandatory 72-hour execution timelock.
- **Emergency Circuit (Hot-Patch):** A predefined **5-of-9** threshold can enact **VK-only** emergency patches (e.g., ZK-Bridge VK update).
  - **Key Allocation and Independence (Normative):** The 9 keys are strictly allocated: Core Team (3), Independent Security Auditor (3 distinct entities), Community/Validator Rep (3). The 5-of-9 threshold MUST include at least one valid signature from each group.
  - **Sunset Clause (Normative):** Emergency patches automatically expire 14 days after activation unless ratified by a full DAO vote. The emergency patch mechanism MUST NOT be capable of modifying the Sunset Clause duration.

9.3 Freeze Points

The cryptographic specification and the tokenomics parameters are hash-pinned and frozen prior to external audits and the formal DAO launch.

10. Roadmap and KPIs

10.1 Phased Rollout

1. **MVP SDK (Rust/TS):** (Completed 2025-09)
2. **DAO Launch & Tokenomics Freeze:** (2025-11)
3. **Public Testnet-0 (L1 DA Chain):** (2026-01) - PoUD+PoDE, Nil-VRF, basic economics.
4. **Edge-Swarm Beta (Retrieval Economy):** (2026-04) - Mobile client, $BW activated, QoS Oracle.
5. **Rollup Bridge Mainnet (L2 Settlement):** (2026-06) - EVM L2 integration, ZK-Bridge, Deal NFTs.
6. **Mainnet-1:** (2026-09).

10.2 Key Performance Indicators (Targets)

Metric	Target
Onboarding Time (Plaintext)	Minutes (CPU-bound)
Epoch Proof Size (Aggregated)	1.2 kB (post-recursion)
Retrieval RTT (p95)	400 ms (across 5 geo regions)
On-chain Verify Gas (L2 Bridge)	120k Gas
Durability	11 nines (modeled)
Sampling FP/FN rate	0.5% / 0.1% (monthly audit)
VRF Beacon Verification Gas (L1)	97 k Gas (1 pairing)

Appendix A: Core Cryptographic Primitives and Parameters

This appendix details the normative cryptographic primitives, parameters, and policies used in NilStore (Core v2.0).

A.1 Dial Parameters (Baseline Profile "S-512")

A **dial profile** defines the core cryptographic parameters and the Proof-of-Delayed-Encode (PoDE) settings.

Symbol	Description	Baseline "S-512"
`Curve`	Elliptic Curve (for KZG and VRF)	**BLS12-381** (Mandatory)
`r`	BLS12-381 subgroup order	`0x73EDA753299D7D483339D80809A1D8055`
`H_t`	PoDE Argon2id time cost (iterations)	3 (Calibrated for Δ_work=1s)

Symbol	Description	Baseline "S-512"
`H_m`	PoDE Argon2id memory cost (KiB)	1048576 (1 GiB)
`H_p`	PoDE Argon2id parallelism	1 (Mandatory Sequential)

A.2 Domain Identifiers and Tags

`DomainID : u16` partitions digests by purpose. Digests are computed as: `digest = Blake2s-256( Version ‖ DomainID ‖ payload )`

Core domain strings used with Blake2s-256 across modules:

Tag	Purpose
`"NIL_VRF_OUT"`	VRF output compression
`"BLS12381G2_XMD:SHA-256_SSWU_RO_NIL_VRF_H2G1"`	VRF hash_to_G2 DST
`"NIL_BEACON"`	Epoch beacon derivation from VRF output
`"NilStore-Sample"`	Retrieval-sampling seed from epoch beacon
`"PODE_DERIVE_ARGON_V1"`	PoDE Derive function tag
`"PODE_INPUT_DIGEST_V1"`	PoDE input data hashing
`"BATMAN-SHARE"`	Deterministic share-selection label
`"NILSTORE-SALT-V1"`	Placement Anti-Grinding Salt derivation

A.3 File Manifest & Crypto Policy (Normative)

NilStore uses a content-addressed file manifest and a deterministic encryption policy.

- **Root CID** = `Blake2s-256("FILE-MANIFEST-V1" || CanonicalCBOR(manifest))`.
- **File Master Key (FMK)** (32B) is HPKE-wrapped to authorized retrieval keys (`"FMK-WRAP-HPKE-V1"`).
- **Deterministic Key/Nonce Derivation (Normative):** Per-DU Content Encryption Keys (CEKs) and Nonces are derived deterministically from the FMK. `(CEK_32B, Nonce_12B) = HKDF-SHA256(IKM=FMK, info="DU-KEYS-V1" || du_id, L=44)`.
- **AEAD: AES-256-GCM** using the derived CEK and the deterministic 96-bit Nonce.
- **Security Warning (Nonce Reuse):** This deterministic derivation is secure ONLY under the strict assumption that DUs are immutable (Write-Once) and that `du_id` is unique for every distinct plaintext under the same FMK.
- **DU CID** = `Blake2s-256("DU-CID-V1" || ciphertext||tag)`.

Appendix B: Threat & Abuse Scenarios and Mitigations

Scenario	Attack surface	Detect / Prevent (Design)	Normative anchor(s)
Wash-retrieval / Self-dealing	SP scripts fake clients to inflate bandwidth usage	Risk-based sampling (Abuse Score); Challenge-nonce + expiry in receipts; watchers verify Ed25519; Poseidon receipt root commitment; RTT Oracle verification (> network floor).	§6.6 (Receipts/Sampling), §4.2 (QoS Oracle)
RTT Oracle collusion	Gateways/attesters collude to post low RTT	Stake-weighted attesters (quadratic weighting, influence cap); challenge-response tokens; mandatory VDF proof in RTT window; ASN/region diversity; randomized assignments; slashable fraud proofs.	§4.2 (RTT Oracle)
Commitment drift in repair	Repaired shards bound to a *new* commitment	Repaired shards MUST open against the **original DU KZG** commitment (`C_root`); reject new commitments.	§3.3 (Autonomous Repair)
Beacon Grinding	Aggregator selects subset of VRF shares to bias beacon	Deterministic Share Selection: Canonical set defined by lowest scores derived from previous epoch's beacon. Aggregator cannot choose the input set.	§6.4.3 (Nil-VRF)
Bridge/rollup trust	VK swap or replay of old epoch	L2 bridge pins `vk_hash`; monotone `epoch_id`; timelocked VK upgrades; 5-of-9 Emergency Circuit with strict role diversity and sunset clause.	§2.4 (ZK-Bridge), §9.2 (Governance)

Scenario	Attack surface	Detect / Prevent (Design)	Normative anchor(s)
Lattice capture (Sybil/cartel)	SPs concentrate shards topologically	One-shard-per-SP-per-cell; minimum cell distance; Secure Identity Binding (Stake_Min_Cell + VDF proof required to move/register coordinates).	§3.2.1 (Placement), §4.1.1 (Identity Binding)
Shard withholding (availability)	SP stores but doesn't serve (or drops data)	Vesting tied to valid PoUD + PoDE; slashing for missed epochs (quadratic + correlation factor); Anti-withholding penalties (immediate penalty > repair bounty).	§7.3 (Vesting/Slashing), §3.3 (Anti-withholding)
PoDE Pre-computation	SP computes derivations before the challenge window	`Derive` function input is salted with the epoch beacon (unpredictable); Argon2id enforces sequentiality (H_p=1); Watchers enforce timing digests within `Δ_submit`.	§6.3 (PoDE), §6.4 (Nil-VRF)