NilStore Litepaper: The Sealing-Free Storage Network

Contents

NilStore Litepaper: The Sealing-Free Storage Network **2**
1. Introduction & Value Proposition 2
    Value for Storage Providers 2
    Value for Data Owners 2
    Data Layout: Erasure Coding & Sharding 2
2. The Core Innovation: PoUD + PoDE 3
    A. Proof-of-Useful-Data (PoUD) 3
    B. Proof-of-Delayed-Encode (PoDE) 3
3. The Architecture: A Hybrid Approach 4
    Layer 1: Consensus & Verification (Cosmos-SDK) 4
    Layer 2: Settlement & Governance (EVM ZK-Rollup) 4
4. The Lifecycle of a File 4
    Step 1: Ingestion (Client-Side) 4
    Step 2: Storage (Provider-Side) 4
    Step 3: Verification (The Epoch Loop) 4
    Step 4: Retrieval & Scaling 4
5. The Living Network: Churn, Exits, & Auto-Healing 5
    A. Entering & Leaving 5
    B. Handling Failure (Churn) 5
    C. Auto-Healing (Bounty Mechanism) 5
6. The Economy ($STOR-Only) 5
7. Why Developers Care 6

NilStore Litepaper: The Sealing-Free Storage Network

Technical Overview v1.1

1. Introduction & Value Proposition

NilStore is a high-throughput, verifiable decentralized storage network designed to democratize access to the storage economy while delivering cloud-grade performance.

By utilizing a novel **Proof-of-Useful-Data (PoUD)** and **Proof-of-Delayed-Encode (PoDE)** consensus, NilStore enables a diverse marketplace of Storage Providers (SPs) running on commodity hardware to provide instant, verifiable data retrieval. This architecture ensures data is always available for high-performance workloads without the latency or hardware overhead of legacy decentralized protocols.

Value for Storage Providers

- **Commodity Hardware Access:** The protocol relies on memory-hard timing checks (Argon2id) rather than GPU-intensive sealing. Providers can participate with standard enterprise CPUs and NVMe/HDD storage.
- **Fluid Market Entry & Exit:** Providers are not locked into multi-year commitments. Through a **Dynamic Friction** model, SPs can exit the network efficiently when capacity is ample, while the protocol economically disincentivizes exit only during periods of scarcity.
- **Dual Revenue Streams:** Providers earn **$STOR** for committed capacity and additional dynamic fees for serving bandwidth, aligning incentives with network performance.

Value for Data Owners

- **Instant Availability:** Data is stored in a retrieval-ready format (canonical bytes). There is no "unsealing" latency, enabling high-performance workloads.
- **Native CDN Scaling:** The protocol automatically detects high-demand content ("Heat Index") and spawns temporary **"Hot Replicas"** across additional providers. This provides built-in, elastic scaling without manual configuration.
- **Configurable Resilience:** Users define durability targets (e.g., "11 nines"), which the protocol maps to specific redundancy profiles.

Data Layout: Erasure Coding & Sharding

NilStore achieves durability and performance through precise data fragmentation, not simple replication.

- **Data Units (DUs):** Files are packed into standardized chunks called Data Units.
- **Configurable Erasure Coding:** Each DU is mathematically split using Reed-Solomon encoding (RS).
  - **The "Knob":** Users can configure the redundancy level **per file**. You set a target (e.g., "Standard", "Archive", or "Mission Critical"), and the system automatically selects the optimal ratio of data shards to parity shards.

    - **Example:** A "Standard" profile might split data into 12 shards where any 9 are needed.
    - **"The Tank":** A "Mission Critical" profile could use an extreme ratio like RS(30,10). This means 30 shards are stored, but only 10 are needed.
        - **Significant Durability:** You could lose 20 entire nodes (2/3rds of the network) and your data is safe.
        - **Blazing Speed:** You download from the fastest 10 nodes out of 30. This acts like a **High-Performance CDN**, aggressively ignoring stragglers and maximizing throughput.
- **Distribution:** These shards are distributed to distinct nodes across the network topology.

Why this matters: * **Customizable Resilience:** You choose the safety level. A family photo album can be set to "Archive" (tolerating many node failures), while a temporary cache file can use "Standard" (lower cost). * **Parallel Throughput:** Regardless of the profile, the client software automatically downloads from the fastest available subset of nodes simultaneously. This "race to download" ensures you are never bottlenecked by the slowest server. * **Efficient Repair:** If a node fails, the network mathematically reconstructs just the specific missing shard using the remaining healthy ones, without needing to move the full file.

---

2. The Core Innovation: PoUD + PoDE

Instead of one massive proof, NilStore uses two lightweight, interacting proofs every epoch (24 hours).

A. Proof-of-Useful-Data (PoUD)

- **The Concept:** Cryptographic verification of physical data possession.
- **The Mechanism: KZG Commitments.**
- **Process:** When a file is uploaded, it is cryptographically "committed" to a mathematical root (`C_root`). Every epoch, the network challenges the Storage Provider (SP) to open a random specific byte of that file. Using KZG properties, the SP proves possession of that exact byte relative to the `C_root` with a compact 48-byte proof.

B. Proof-of-Delayed-Encode (PoDE)

- **The Concept:** Verification of local storage latency (preventing on-demand fetching).
- **The Mechanism: Argon2id** (Memory-Hard Function).
- **Process:** The network sends a random "salt" (from the Epoch Beacon). The SP must perform a precise, memory-heavy calculation on a chunk of their data.
- **Security:** This calculation is tuned to take exactly **1 second** on a reference CPU. If the SP attempts to download the data from a remote source *before* running the calculation, the network latency (e.g., 200ms) plus the calculation time (1s) will cause them to miss the strict submission deadline. Success proves the data is stored locally.

---

3. The Architecture: A Hybrid Approach

NilStore splits the workload to optimize for speed, cost, and composability.

Layer 1: Consensus & Verification (Cosmos-SDK)

- **Role:** The "Math Police."
- **Function:** Optimized strictly for high-throughput verification. It does **not** run general-purpose smart contracts. It verifies KZG/PoDE proofs and produces the **Nil-VRF Epoch Beacon** (unbiased randomness) to ensure fair challenges.

Layer 2: Settlement & Governance (EVM ZK-Rollup)

- **Role:** The "Bank."
- **Function:** Hosts the **$STOR** token, payment logic, staking contracts, and the DAO.
- **The Bridge:** The L1 aggregates millions of verification results into a single, succinct **Zero-Knowledge Proof** which is posted to L2. This proof acts as the trigger for settlement: *"The L1 consensus attests that these miners performed their duties."*

---

4. The Lifecycle of a File

Step 1: Ingestion (Client-Side)

1. **Input:** The user provides a file to the NilStore SDK.
2. **Packing:** The file is segmented into **Data Units (DUs)**.
3. **Encoding:** DUs are Erasure Coded (e.g., 12 shards).
4. **Placement:** The protocol determines the deterministic placement for each shard using the **Nil-Lattice** topology, ensuring geographic distribution.

Step 2: Storage (Provider-Side)

1. **Uptake:** The SP downloads their assigned shard.
2. **Storage:** The shard is stored as **Plaintext** (or ciphertext per privacy settings), ready for immediate access.
3. **Bonding:** The SP bonds **$STOR** tokens as collateral against data loss.

Step 3: Verification (The Epoch Loop)

1. **Beacon:** The network generates the verifiable random beacon (Nil-VRF).
2. **Challenge:** Specific byte offsets are selected for verification.
3. **Response:** The SP computes the KZG proof and PoDE timing check.
4. **Settlement:** Valid proofs accrue rewards; missing proofs trigger slashing (collateral burn).

Step 4: Retrieval & Scaling

1. **Discovery:** The **Nil-Mesh** routing layer identifies the fastest available nodes holding the required shards.

2. **Viral Scaling:** If a file's access pattern spikes ("High Heat"), the network automatically spawns "**Hot Replicas**" on high-bandwidth nodes to meet demand.
3. **Payment:** The user streams **$STOR** payments (micro-transactions) for bandwidth consumed.
4. **Download:** Data is streamed in parallel from the optimal set of providers.

---

5. The Living Network: Churn, Exits, & Auto-Healing

NilStore is designed as an adaptive system that autonomously manages health and capacity.

A. Entering & Leaving

- **Entry:** Permissionless. Any provider can bond $STOR and join the network.
- **Exit (Dynamic Friction):** Exits are governed by network health.
  - **Surplus Capacity:** Exits are fast (~24h) and low-cost.
  - **Scarcity:** Exits are delayed (up to 7 days) with higher fees, ensuring stability and preventing "capital flight" during stress events.

B. Handling Failure (Churn)

- **Detection:** Missed proofs trigger immediate penalties.
- **The Correlation Factor:** Slashing penalties are non-linear. A single node failure incurs a nominal fine. However, if many nodes fail simultaneously within the same failure domain (ASN/Region), penalties multiply (up to 5x). This effectively punishes centralized infrastructure risks.

C. Auto-Healing (Bounty Mechanism)

- **Trigger:** If shard availability drops below the safety threshold (e.g., 10/12 shards remaining), the network posts a **Repair Bounty**.
- **Action:** Any node on the network can reconstruct the missing shard and submit it to the L1.
- **Reward:** The repairer is paid from the deal's escrow, while the failing node incurs a penalty exceeding the repair cost.

---

6. The Economy ($STOR-Only)

The network operates as a self-contained economy without reliance on external stablecoins or oracles.

- **$STOR Token:** The single medium for Staking (Security) and Bandwidth (Utility).
- **Burn Mechanism:** A portion of every retrieval fee is **burned** (removed from supply), functionally similar to Ethereum's EIP-1559. This creates a deflationary pressure correlated with network usage.
- **Real Pricing:** Storage and bandwidth are priced by the market to reflect physical infrastructure costs, preventing unsustainable subsidy models.

---

7. Why Developers Care

1. **S3 Compatibility:** The SDK abstracts the complexity, offering a familiar object storage interface.
2. **Verifiability:** Cryptographic proofs provide certainty of data persistence.
3. **Performance:** The lack of “sealing” enables low-latency, high-throughput retrieval.
4. **Resilience:** The **Nil-Lattice** placement guarantees topological diversity, ensuring data survives regional outages.