DSM Storage Nodes — Protobuf-Only Implementation Specification

Deterministic, Clockless, Signature-Free, Consensus-Free (Envelope v3; Schema 2.4.0)

Brandon “Cryptskii” Ramsay

October 27, 2025

Contents

Abstract

This document is the *normative* implementation specification for DSM storage nodes under Envelope v3 and Schema 2.4.0. Storage nodes are *clockless* (no timestamps) and *signature-free*: they store and mirror bytes, enforce address and byte accounting invariants, and never attest. All verifiability is client-side via domain-separated BLAKE3-256 over deterministically encoded Protocol Buffers (protobuf-only; DSM-CPE), with device-side SPHINCS+ signatures where required. The spec defines deterministic acceptance predicates, message schemas, keyed-permutation replica placement, capacity signaling, registry update rules, PaidK spend gate, stake exit via DrainProof, and—new in this revision—*contact anchoring*, *token policy (CTPA) anchoring*, and *DLV (Deterministic Limbo Vault) anchoring*, while banning JSON/CBOR/hex/base64 paths.

Normative Language

Normative

MUST, *MUST NOT*, *SHOULD*, *MAY* are used per RFC 2119. All predicates are clockless; cycle indices are counters embedded in accepted objects, never wall time.

1 Purpose and Philosophy

DSM maximizes determinism and minimizes trust in infrastructure:

- No validators, sequencers, or leaders.
- Dumb storage nodes: store, mirror, and enforce arithmetic; never sign.
- No clocks in acceptance; predicates are over domain-separated digests and SMT roots.
- Deterministic placement and repair by keyed permutation over the active registry.
- Capacity is math-driven via mirrored utilization signals; no VRFs, votes, or auctions.

2 Notation and Primitives

Hash. $\mathsf{H} := \mathsf{BLAKE3\text{-}256}$.

KDF. HKDF for key stepping.

Device signatures. SPHINCS+ (post-quantum). *Nodes never sign.*

Optional secrecy. ML-KEM with deterministic HKDF.

Domain separation. ASCII tag with NUL, e.g. `DSM/object\0`, `DSM/place\0`, `DSM/registry\0`, `DSM/bytecommit\0`, `DSM/signal/up\0`, `DSM/signal/down\0`, `DSM/pay/storage\0`, `DSM/genesis\0`, `DSM/policy\0`, `DSM/policy/anchor\0`, `DSM/dlv/create\0`, `DSM/dlv/open\0`, `DSM/contact/add\0`, `DSM/contact/accept\0`. All hashes are over *domain* $\|$ *deterministic-protobuf-bytes*.

3 Deterministic Protobuf Encoding (DSM-CPE)

DSM replaces CBOR with *deterministic* Protocol Buffers (proto3 syntax). Implementations *MUST* satisfy:

1. **Deterministic serialization.** Serialize messages with deterministic ordering: fields strictly ascending by field number; map entries sorted by key lexicographically as raw bytes; repeated fields serialized in their in-message order. Varints *MUST* use the unique shortest form; no trailing zeros.
2. **Presence discipline.** Fields marked `required` (or proto3 `optional`) *MUST* be present on the wire. Unknown fields *MUST NOT* appear. Default-zero fields intended to be set *MUST* be present using `optional`.
3. **No alternative encodings.** No JSON, no base64, no hex, no CBOR. I/O is protobuf-only.
4. **Digest rule.** For any message M, $\mathsf{H}(\texttt{name\textbackslash 0}\|\text{ProtoDet}(M))$ where ProtoDet is the DSM-CPE serialization.

4 Device Identity, Binding, and Step Keys

Device ID. $\text{DevID}_A = \mathsf{H}(\texttt{DSM/device\textbackslash 0}\|pk_A\|\text{attest})$.

DBRW binding. $K_{\text{bind}} = \mathsf{H}(\texttt{DSM/DBRW\textbackslash 0}\|\mathsf{H}(\text{HW\_entropy}\|\text{ENV\_fp}))$.

Step keys. derive via HKDF with salts tied to the last accepted device commit digest, never wall time. Only devices sign stitched receipts; nodes do not sign.

5 Genesis Anchor (Commit–Reveal, Clockless)

A single network-wide genesis anchor binds parameters, initial registry, and entropy. Let

$$G = \mathsf{H}(\texttt{DSM/genesis\0} \| \mathrm{ProtoDet}(A_0)),$$
$$s_0 = \mathsf{H}(\texttt{DSM/step-salt\0} \| G).$$

Commit–reveal produces $\eta_0 = \mathsf{H}(\texttt{DSM/anchor/eta\0} \| D_{\mathrm{commit}} \| D_{\mathrm{reveal}})$; withholding a reveal cannot bias ordering post-commit.

Genesis Parameters (normative).

$P = [\mathsf{H} = \mathsf{BLAKE3\text{-}256},\ \mathrm{Sig} = \mathsf{SPHINCS+},\ \mathrm{KDF} = \mathsf{HKDF},\ N,\ K,\ U_{\uparrow},\ U_{\downarrow},\ w,\ G_{\mathrm{new}},\ \mathrm{FLAT\_RATE}].$

6 Addressing, Placement, and Enforcement

Object address. For node-local partition $\mathrm{DLV}_{\mathrm{node}}$, path, and content:

$$\mathsf{addr} = \mathsf{H}(\texttt{DSM/object\0} \| \mathrm{DLV}_{\mathrm{node}} \| \mathrm{path} \| \mathsf{H}(\mathrm{content})).$$

Replica set. Given active registry vector $\mathcal{N}$ (Section 9), redundancy (N, K):

$$\mathrm{replicas}(\mathsf{addr}) = \text{first } N \text{ entries of } \mathrm{Permute}(\mathsf{H}(\texttt{DSM/place\0} \| \mathsf{addr}),\ \mathcal{N})\,.$$

PUT enforcement. Nodes *MUST* reject if partition mismatch, address mismatch, or byte accounting would exceed capacity; otherwise store. GET returns bytes; clients re-verify addr from bytes.

7 Node Storage SMT and ByteCommit Mirroring

Each node maintains a sparse Merkle tree (SMT) over served object addresses. Leaves encode $[\mathsf{addr}, \mathsf{H}(\mathrm{content}), \mathrm{len}]$ under domain DSM/smt-node\0. Paths are MSB-first; ZERO_LEAF $= 32 \times 0x00$. After applying all PUT/DELETEs for cycle t:

$$R_t^{\mathrm{node}} = \mathrm{SMTRoot}(\mathrm{NodeStorage}_t), \qquad \mathsf{bytes}_t^{\mathrm{used}} = \sum_{\ell \in \mathrm{leaves}_t} \mathrm{len}(\ell).$$

ByteCommit Message (Envelope v3)

Listing 1: ByteCommitV3 message

```
message ByteCommitV3 {
  // domain for digest: "DSM/bytecommit\0"
```

```
  bytes node_id = 1; // 32 bytes
  uint64 cycle_index = 2; // t
  bytes smt_root = 3; // R_t^node (32 bytes)
  uint64 bytes_used = 4; // total used in partition
  bytes parent_digest = 5; // H(B_{t-1}); H(B_0) = 32 x 0x00
}
```

Let B_t be the DSM-CPE bytes; $H(B_t) := \mathsf{H}(\texttt{DSM/bytecommit\0} \| B_t)$. Mirror B_t like ordinary content with address

$$\mathsf{addr}_t^B = \mathsf{H}\big(\texttt{DSM/obj-bytecommit\0} \| \text{node\_id} \| t \| H(B_t)\big).$$

A verifier accepts B_t iff: deterministic protobuf, domain tag matches, digest matches, parent link valid (for $t > 0$), SMT root validates, and $\geq q$ identical mirrored copies are fetched.

8 Capacity Signals and Position Delta

Over a window w of consecutive cycles, define utilization $u_j = \mathsf{bytes}_j^{\text{used}}/C$ for capacity C.

Up/Down Signal Messages

Listing 2: UpSignalV3 and DownSignalV3

```
message UpSignalV3 {
  // domain: "DSM/signal/up\0"
  bytes node_id = 1;
  uint64 capacity = 2; // C
  repeated bytes anchors = 3; // H(B_{t-w+1}),...,H(B_t)
}
message DownSignalV3 {
  // domain: "DSM/signal/down\0"
  bytes node_id = 1;
  uint64 capacity = 2;
  repeated bytes anchors = 3;
}
```

Validity. Verifier fetches each referenced $H(B_j)$ and checks $u_j \geq U_\uparrow$ (Up) or $u_j \leq U_\downarrow$ (Down) for every j in the window. Let U and D be the discovered valid signals over discovery window W (in cycle indices). The position delta is $\Delta P = |U| - |D|$.

9 Registry, Applicants, and Deterministic Ordering

Registry Message

Listing 3: RegistryV3

```
message RegistryV3 {
```

```
  // domain: "DSM/registry\0"
  repeated bytes node_ids = 1; // sorted ascending lex order
}
```

Let $\mathcal{N} = [\text{node\_id}_0, \ldots, \text{node\_id}_{n-1}]$; address $\mathsf{addr}_{\text{reg}} = \mathsf{H}(\texttt{DSM/registry\0} \| \text{ProtoDet}(\mathsf{RegistryV3}))$. Clients use this as the placement source of truth.

Applicant Pack and Ranking (No VRF, No Leader)

Listing 4: ApplicantV3

```
message ApplicantV3 {
  // domain: "DSM/apply\0"
  bytes hw_bind = 1; // DBRW bind
  bytes env_fp = 2; // environment fingerprint
  uint64 capacity = 3; // declared C
  bytes stake_dlv = 4; // StakeDLV address
  bytes seed_app = 5; // H("DSM/app_seed\0" || hw_bind || G)
}
```

Deterministic salt:

$$\text{salt} = \mathsf{H}\big(\texttt{DSM/positions/salt\0} \| G \| \mathsf{addr}_{\text{reg}} \| \mathsf{H}(\text{concat}(U)) \| \mathsf{H}(\text{concat}(D))\big).$$

Rank: $\text{rank}(\mathsf{addr}_{\text{apply}}) = \mathsf{H}(\texttt{DSM/order\0} \| \text{salt} \| \text{seed\_app})$. The $|\Delta P|$ winners are lowest ranks; add on $\Delta P > 0$. On $\Delta P < 0$, prune the $|\Delta P|$ lowest-utilization nodes deterministically with tiebreaker by `node_id`. The map $\mathcal{N} \to \mathcal{N}'$ is a pure function of public inputs.

10 New-Operator Grace (Cycle-Based)

A new node is grace-protected for G_{new} cycles beginning at its first published B_{t_0} after activation. During grace, Down signals from that node are ignored for ΔP and pruning. Verification relies only on mirrored B-chain indices.

11 Repair, Mid-Cycle Replacement, Continuity

Reads succeed with any K replies; clients always verify `addr` from bytes. On prune or voluntary exit, clients extend Permute$(\cdot)$ to restore N replicas. Batching writes is recommended; acceptance is hash-only so client-written repairs are equivalent to operator-written.

12 Contacts: Creation, Anchoring, and Storage Behavior

Goal. Establish a verifiable contact edge between two devices without clocks, sequencers, or server attestations. Storage nodes *only* mirror the bytes; all authenticity is device-side.

Contact Add / Accept Messages

Listing 5: ContactAddV3 and ContactAcceptV3

```
message ContactAddV3 {
  // domain: "DSM/contact/add\0"
  bytes author_device_id = 1; // 32 bytes
  bytes contact_device_id = 2; // 32 bytes
  bytes contact_chain_tip = 3; // 32 bytes (peer-provided tip for this relationship or
      identity)
  bytes parent_digest = 4; // chain link of the author's local contact stream (optional)
}

message ContactAcceptV3 {
  // domain: "DSM/contact/accept\0"
  bytes accepter_device_id = 1; // 32 bytes
  bytes add_digest = 2; // H(DSM/contact/add\0 || ProtoDet(ContactAddV3))
  bytes local_chain_tip = 3; // 32 bytes (accepter's tip for this relationship or identity)
}
```

Addressing (contact namespace). Contacts are mirrored under paths of the form

$$\texttt{contacts/}\langle \text{author\_device\_id} \rangle / \langle \mathsf{H}(\text{ProtoDet}(\cdot)) \rangle,$$

but the address is always computed by *domain* ∥ *bytes* per §**??**, so any path-only manipulation is irrelevant to verifiers.

Verifier acceptance (normative).

C1. ContactAddV3. Accept iff DSM-CPE encoding and domain tag match, and a valid device-side SPHINCS+ signature (external envelope) authenticates $\mathsf{H}(\texttt{DSM/contact/add\0} \| \text{ProtoDet}(\cdot))$. Nodes *do not* verify signatures.

C2. ContactAcceptV3. Accept iff the referenced add_digest is accepted under **C1** and the accepter's device envelope signs the *accept* digest. If present, matching tips are cached for future bilateral sessions.

Notes. Contact creation is unilateral and sufficient for discovery; `ContactAcceptV3` tightens the edge with bilateral evidence. Genesis hashes are only required at anchoring/trust bootstrap; thereafter the `chain_tip` uniquely distinguishes the relationship-local straight chain.

13 Token Policies (CTPA): Creation and Anchoring

Goal. Publish an immutable, content-addressed token policy (CTPA) that devices MUST verify prior to token acceptance or transfer. Nodes mirror bytes; all enforcement occurs on devices at state-transition time.

Policy and Anchor Messages

Listing 6: TokenPolicyV3 and PolicyAnchorV3

```
message TokenPolicyV3 {
  // domain: "DSM/policy\0"
  bytes policy_bytes = 1; // canonical policy program/content (opaque to nodes)
}

message PolicyAnchorV3 {
  // domain: "DSM/policy/anchor\0"
  bytes policy_digest = 1; // H(DSM/policy\0 || ProtoDet(TokenPolicyV3))
  bytes author_device_id = 2; // 32 bytes
  bytes parent_digest = 3; // author's policy stream link (optional)
}
```

Addressing (policy namespace). Policies are mirrored under `policy/`⟨`policy_digest`⟩; anchors under `policy/anchor/`⟨`policy_digest`⟩. Addressing correctness is enforced by $\mathsf{addr} = \mathsf{H}(\texttt{DSM/object\0} \| \mathrm{DLV}_{\mathrm{node}} \| \mathrm{pa}$

Verifier acceptance (normative).

P1. TokenPolicyV3. Accept iff DSM-CPE encoding and domain tag match; compute $p = \mathsf{H}(\texttt{DSM/policy\0} \| \mathrm{ProtoDet}(\cdot))$.

P2. PolicyAnchorV3. Accept iff DSM-CPE encoding and domain tag match, `policy_digest` equals p from **P1**, and the author's device envelope signs the anchor digest. Devices MUST cache p and verify policy compliance at every relevant token state update. Nodes never interpret policy content.

CTPA enforcement (device-side). Before accepting or producing any token state that references p, devices MUST confirm the on-device cache contains the exact bytes for p and that the transition satisfies the policy program. This is a *hard* precondition; caching is content-addressed and safe to distribute via any mirror.

14 Deterministic Limbo Vaults (DLV): Creation and Opening

Goal. Anchor vaults whose keys are derived only upon meeting deterministic conditions; no timestamps, no third-party authority. Nodes mirror bytes; verifiers and authors prove condition satisfaction.

DLV Messages

Listing 7: DLVCreateV3 and DLVOpenV3

```
message DLVCreateV3 {
```

```
  // domain: "DSM/dlv/create\0"
  bytes device_id = 1; // 32 bytes
  bytes policy_digest = 2; // CTPA policy governing this vault
  bytes precommit = 3; // H(DSM/dlv/precommit\0 || reveal_material)
  bytes vault_id = 4; // H(DSM/dlv\0 || device_id || policy_digest || precommit)
  bytes parent_digest = 5; // author's vault stream link (optional)
}

message DLVOpenV3 {
  // domain: "DSM/dlv/open\0"
  bytes device_id = 1; // 32 bytes
  bytes vault_id = 2; // must match an accepted DLVCreateV3
  bytes reveal_material = 3; // proves H(preimage) == precommit
}
```

Addressing (DLV namespace). `dlv/⟨vault_id⟩/create` for `DLVCreateV3`; `dlv/⟨vault_id⟩/open/⟨H(`ProtoDet(
for openings. As always, byte-level addressing dominates path conventions.

Verifier acceptance (normative).

V1. DLVCreateV3. Accept iff DSM-CPE encoding and domain tag match; compute $v = \mathsf{H}(\texttt{DSM/dlv\0} \| \text{device\_id} \| \text{policy\_digest} \| \text{precommit})$ and confirm `vault_id` equals v. Device envelope must sign the create digest.

V2. DLVOpenV3. Accept iff DSM-CPE encoding and domain tag match, $H($`DSM/dlv/precommit\0`$\|$reveal_materia precommit from the accepted create for this `vault_id`, and the opening signature verifies on the device envelope. Additional policy conditions (e.g., multi-branch forks, pre-commit forking) MUST be checked against the cached CTPA p referenced by `policy_digest`.

Relationship to storage nodes. Nodes do not gate, unlock, or attest vaults. They only store the create/open artifacts and account for bytes. Unlock conditions are enforced entirely by verifiers reading mirrored evidence.

15 Stake DLV and Signature-Free Exit

Stake Lock

$$\text{StakeDLV} = \mathsf{H}(\texttt{DSM/stake\0} \| \text{node\_id} \| S \| \text{policy}).$$

Exit Conditions

Stake unlocks if and only if a valid *DrainProof* is mirrored: d consecutive accepted ByteCommits with $\mathsf{bytes}_t^{\text{used}} = 0$ (normative $d = 2$). If prune removed the node, publish the matching DownSignal and DrainProof. No attestations or signatures by nodes are required.

16 PaidK Spend-Gate (Clockless)

A device is receive-only after genesis until it pays any K distinct storage operators a flat fee. Let R be the set of device-signed payment receipts:

$$\text{PaidK}(G, \text{DevID}, R) := \left|\{\, r \in R \mid \text{VerifyPayment}(r) \wedge \text{amt}(r) \geq \text{FLAT\_RATE} \,\}\right| \geq K.$$

On first satisfaction, spend is permanently enabled; no renewals by time.

17 Permutation Determinism

Keyed Fisher–Yates over a stable pre-order (bytewise ascending order of 32-byte digests). Seed: $s = \mathsf{H}(\texttt{DSM/place\textbackslash 0} \| x)$ for item x. Derive swap stream $r_i = \mathsf{H}(s \| \text{uint32}(i))$. Inputs *MUST* be strictly in the stable pre-order before permutation. This ensures all honest clients compute identical replica sets.

18 Acceptance Rules (Normative)

A verifier *MUST* apply the following rules:

1. **Object bytes.** Accept bytes for `addr`only if the response recomputes `addr`exactly.
2. **ByteCommit.** Accept B_t iff DSM-CPE deterministic encoding, correct domain, $H(B_t)$ matches, $t{>}0 \Rightarrow H(B_{t-1})$ matches, SMT root validates (domain tags, ZERO_LEAF, MSB walk), and the object appears identically at $\geq q$ mirrors.
3. **Signals.** Accept Up/Down iff each referenced $H(B_j)$ is accepted and the window inequality holds for all j.
4. **Registry update.** Given prior $\mathcal{N}$, valid U, D, and applicant set A, compute ΔP, compute salt, rank applicants deterministically, update $\mathcal{N}$ by the pure add/remove rules.
5. **PaidK.** Spend enabled iff $\text{PaidK}(G, \text{DevID}, R)$ holds over canonical device-signed receipts.
6. **Contacts.** Accept `ContactAddV3` and `ContactAcceptV3` per §12 rules **C1**–**C2**. Cache tips for bilateral sessions.
7. **Policies (CTPA).** Accept `TokenPolicyV3` and `PolicyAnchorV3` per §13 rules **P1**–**P2**. Reject any token transition that lacks a cached exact-byte policy match.
8. **DLVs.** Accept `DLVCreateV3`/`DLVOpenV3` per §14 rules **V1**–**V2**. Policy satisfaction is mandatory for openings.

19 Parameter Defaults (Normative)

- Redundancy: $N{=}6$, $K{=}3$.

- Signal thresholds: $U_{\uparrow}$=0.85, $U_{\downarrow}$=0.35, window w=3, discovery window W=4 (cycles).
- New-operator grace: $G_{\text{new}} \in \{5, 8, 12\}$ cycles (typical 8).
- DrainProof length: d=2.
- PaidK: K=3 distinct operators; `FLAT_RATE` deployment-defined.

20 Security Rationale (Sketches)

Integrity. Acceptance recomputes addrfrom bytes; wrong bytes cannot pass without a second preimage. **Utilization verifiability.** Signals carry references to accepted B-chain slices; forging requires breaking mirror quorum or hashes. **Deterministic scaling.** Registry updates are pure functions over hashed inputs; all honest verifiers compute the same result. **Availability.** Reads succeed with K live replicas; repairs are deterministic. **Stake unlock safety.** Zero-occupancy is publicly checkable from mirrored B_t; stake unlocks iff emptiness holds for d cycles. **Clocklessness.** All predicates count accepted artifacts and indices; no timestamps are consulted. **Contact/policy/DLV soundness.** Contacts require device signatures and optional bilateral confirmation; CTPA anchors bind immutable policy bytes; DLV precommit/open is hash-equivalence checked and policy-gated, all without node attestations.

Appendix A: Minimal `.proto` Sketch (Informative)

Listing 8: DSM core messages (sketch)

```
syntax = "proto3";
package dsm.v3;

// Deterministic serialization required for all messages.

message ByteCommitV3 {
  bytes node_id = 1; // 32 bytes
  uint64 cycle_index = 2; // t
  bytes smt_root = 3; // 32 bytes
  uint64 bytes_used = 4;
  bytes parent_digest = 5; // 32 bytes; all-zero at t=0
}

message UpSignalV3 {
  bytes node_id = 1;
  uint64 capacity = 2;
  repeated bytes anchors= 3; // H(B_{t-w+1}),...,H(B_t)
}

message DownSignalV3 {
  bytes node_id = 1;
  uint64 capacity = 2;
  repeated bytes anchors= 3;
```

```
}

message RegistryV3 {
  repeated bytes node_ids = 1; // sorted ascending
}

message ApplicantV3 {
  bytes hw_bind = 1;
  bytes env_fp = 2;
  uint64 capacity = 3;
  bytes stake_dlv = 4;
  bytes seed_app = 5; // H("DSM/app_seed\0" || hw_bind || G)
}

// Contacts
message ContactAddV3 {
  bytes author_device_id = 1; // 32 bytes
  bytes contact_device_id = 2; // 32 bytes
  bytes contact_chain_tip = 3; // 32 bytes
  bytes parent_digest = 4; // optional
}
message ContactAcceptV3 {
  bytes accepter_device_id = 1; // 32 bytes
  bytes add_digest = 2; // 32 bytes
  bytes local_chain_tip = 3; // 32 bytes
}

// Token policy (CTPA)
message TokenPolicyV3 {
  bytes policy_bytes = 1; // opaque to nodes
}
message PolicyAnchorV3 {
  bytes policy_digest = 1; // H(DSM/policy\0 || ProtoDet(TokenPolicyV3))
  bytes author_device_id = 2; // 32 bytes
  bytes parent_digest = 3; // optional
}

// DLVs
message DLVCreateV3 {
  bytes device_id = 1; // 32 bytes
  bytes policy_digest = 2; // 32 bytes
  bytes precommit = 3; // 32 bytes
  bytes vault_id = 4; // 32 bytes
  bytes parent_digest = 5; // optional
}
message DLVOpenV3 {
  bytes device_id = 1; // 32 bytes
  bytes vault_id = 2; // 32 bytes
  bytes reveal_material = 3; // variable length
}
```

Informative

Provenance. This protobuf-only spec remains aligned with DSM Envelope v3 and schema v2.4.0. The additions formally specify storage-node treatment (mirror & byte-accounting only) for (i) contact anchoring, (ii) token policy (CTPA) anchoring, and (iii) DLV creation/opening, with all authenticity and enforcement performed client-side via domain-separated hashes and device-side SPHINCS+ signatures.