Annex A — Extended Technical Report

Author: Ahmad Hemmati | Website: https://www.twincodesworld.com

Status: Open Source – Technical Supplement | Version: v1.0

License: Apache 2.0 | Repository: github.com/twincodesworld/LHDNS

Scope & purpose: this annex is the authoritative, implementer-oriented technical reference for LHDNS. It contains canonical serialization rules, crypto bindings, ledger propagation internals, relay & transport details, privacy measures, governance numeric parameters, KPIs, audit formats, test vectors, API endpoints, and threat→mitigation mappings. Implementers should treat values in **DEFAULTS** as bootstrap parameters tunable by on-chain governance.

DEFAULTS / CONSTANTS (reference)

- HASH_ALG = SHA-256
- SIGNATURE_ALG = Ed25519 (64B sig, 32B pub)
- KEX = X25519, HKDF-SHA256
- AEAD = XChaCha20-Poly1305 (fallback AES-GCM if HW accel)
- TIME_WINDOW = 30 seconds
- DEFAULT_TTL = 30 seconds (interactive)
- CLOCK_SKEW = ±60 seconds (accept)
- MAX_ENTRY_SIZE = 4096 bytes
- GOSSIP_FANOUT_DEFAULT = 5
- DIGEST_INTERVAL = 60 seconds
- POW_BASE_BITS = 16 (bootstrap)
- POW_ADAPT_K = 2.0
- STAKE_MIN_VALIDATOR = 10,000 LHD
- STAKE_MIN_RELAY = 100 LHD
- REPLICATION_FACTOR = 3
- DIGEST_RETENTION_DAYS = 30

Module 1 — Cryptographic & Identity Layer

1.1 Goals

Define canonical message formats and deterministic signature inputs; ensure cross-language correctness; define hash-token formula and enc_contact encryption.

1.2 Identifiers

- **service_id**: `svc:sha256:<hex>` — canonical representation; raw bytes used in crypto (32 bytes).

- **client_id**: long-term pseudonymous identifier (public key) stored locally; used for optional long-term bindings.
- **ephemeral key**: per-session ephemeral keypair (Ed25519 for signing, X25519 for KEX).

1.3 Hash-token formula (deterministic)

Recommended deterministic input bytes:

```
input_bytes = LE32(len(eph_pub)) || eph_pub || service_id_bytes ||
service_nonce || LE64(time_window)
hash = SHA256(input_bytes)   # 32 bytes; index token
```

- `eph_pub` = ephemeral public key bytes (raw).
- `service_id_bytes` = 32 raw bytes.
- `service_nonce` = optional per-service nonce (random ≤ 32B), reduces bruteforce.
- `time_window` = floor(ts / TIME_WINDOW).

Rationale: includes eph_pub to increase brute-force cost; service_nonce prevents precompute; time_window bounds lifetime.

1.4 Canonical serialization & signature input (must)

- Use **Canonical CBOR (RFC 8949)** as binary canonical form.
- Implementers using JSON MUST first canonicalize via **JCS (JSON Canonicalization Scheme)** then CBOR-encode the result for signing/AAD.
- **Signature input** is CBOR-canonical bytes of object:

```
{ "hash": "<hex>", "ts": <uint64>, "service_id": <32-byte raw> }
```

- `service_id` must be raw bytes, not ASCII prefixed string.
- The AEAD associated data (AAD) for `enc_contact` is the CBOR canonical serialization of the full entry **with the `enc_contact` field omitted**.

Verification: Verifier reconstructs canonical CBOR and verifies Ed25519 over canonical bytes; any deviation causes rejection.

1.5 enc_contact encryption (concrete KDF + AEAD)

Pseudocode (reference):

```
# Client side: preparing enc_contact
client_x25519_priv, client_x25519_pub = X25519_keypair()
shared = X25519(client_x25519_priv, service_x25519_pub)   # raw 32B
salt = service_id_bytes                                   # 32B
info = b"lhdns/enc_contact/v1"
K = HKDF_SHA256(shared, salt=salt, info=info, length=32)
aad = canonical_cbor(entry_without_enc_contact)
```

```
ciphertext = XChaCha20_Poly1305_Seal(K, plaintext=enc_contact_plain_cbor,
aad=aad)
enc_contact = base64(ciphertext)
```

Service side:

```
shared = X25519(service_x25519_priv, client_x25519_pub)
K = HKDF_SHA256(shared, salt=service_id_bytes, info=b"lhdns/enc_contact/v1",
length=32)
plaintext = XChaCha20_Poly1305_Open(K, ciphertext, aad=aad)
```

- `plaintext` must be canonical CBOR or JCS→CBOR.
- Use XChaCha20-Poly1305 (safe nonce); AES-GCM allowed with careful nonce management & HW accel.

1.6 Entry canonical JSON (canonical structure for ledger)

Canonical entry fields (human-readable view — actual serialized bytes must be canonical CBOR):

```
{
  "version": 1,
  "service_id": "<32B raw or svc:sha256:...>",
  "hash": "0x...",
  "eph_pub": "<base64>",
  "enc_contact": "<base64>",
  "ts": 1690000000,
  "ttl": 30,
  "sig": "<base64>",
  "metadata": { ... }
}
```

- **sig** = Ed25519 signature over the canonical CBOR bytes of object `{ "hash": …, "ts": …, "service_id": … }`.

1.7 Local-node validation

Steps on reception:

1. Validate CBOR schema & `version`.
2. Check `ts` within `CLOCK_SKEW`.
3. Check `ttl` within allowed bounds.
4. Recompute `hash` from eph_pub || service_id || service_nonce || time_window.
5. Verify `sig` with `eph_pub`.
6. Reject malformed/oversize `enc_contact`.
7. Rate-limit by `eph_pub`, circuit or authenticated account; optionally require PoW or micro-fee.
8. If accepted: index by `service_id`, gossip, keep in-memory until `ts + ttl + grace`.

1.8 Error codes (examples)

- `400` — schema error
- `401` — invalid signature
- `402` — fee/PoW required
- `429` — rate-limited
- `410` — expired

1.9 Testing checklist (Module1)

- signature vectors, invalid sig rejection, ttl enforcement, PoW enforcement, enc_contact decrypt stress, replay tests.

Module 2 — Ledger & Propagation (DLN internals)

2.1 Goals

Low-latency ephemeral ledger optimized for short-lived indexed entries (not a permanent blockchain). Prioritize memory/in-memory indexes, digest auditability, and fast gossip.

2.2 Roles

- **Validator nodes:** full validation, gossip, indexing, digest publication.
- **Relay nodes:** forward traffic, may not fully validate.
- **Light/edge nodes:** accept client submissions, perform quick validation.
- **Auditor nodes:** collect signed digests and perform cross-checks (optional).

2.3 Propagation model

- Epidemic gossip (push-pull).
- Default fanout k = `GOSSIP_FANOUT_DEFAULT` (5), adaptive by network size.
- Dedup via per-round Bloom filter.
- Partial redundancy: each entry gossips until TTL expiry or gossip horizon.

2.4 Storage model

- In-memory ephemeral store; no durable storage of full entries (only digests retained).
- Indexing key = `service_id` → list of entries valid within `[ts, ts + ttl]`.
- Grace window default +5s for clock skew.

2.5 Validation on gossip reception

- Schema/version check.
- Verify signature.
- Recompute hash.

- Rate-limit per eph_pub & per peer.
- Index & propagate if valid; penalize peer on invalid.

2.6 Dedup & replay protection

- Each node maintains Bloom filter for recent `hash` values; duplicate suppressed.
- Replay prevented via (hash, eph_pub, ts) tuple uniqueness + bloom.

2.7 Gossip topology & peer sampling

- Unstructured P2P with partial views (Kademlia-like sampling recommended).
- Fanout adaptive: k = clamp(log2(N) * factor, min=3, max=12).
- Backoff under congestion.

2.8 Capacity & pruning

- Node memory cap configurable (e.g., 100k active entries).
- Prune by TTL + grace; maintain dropped-hash Bloom for short period to prevent replay.

2.9 Audit & Digest subsystem (full spec)

2.9.1 Merkle digest construction

- **Leaf** = `SHA256(0x00 || CBOR_canonical(entry_without_enc_contact)).`
- Build binary Merkle tree over sorted leaf hashes (lexicographic).
- **Root** = `SHA256(0x01 || left || right)` recursively.

2.9.2 Signed digest document (format)

```
{
 "node_id": "<b64(node_pub)>",
 "ts": 1690000000,
 "interval_seconds": 60,
 "merkle_root": "0x...",
 "entry_count": 1234,
 "bloom_filter": "<base64>",
 "sig": "<b64(Sign_node(CBOR_canonical(doc_except_sig)))>"
}
```

APIs

- `GET /digest/latest` — returns latest signed digest JSON.
- `POST /proof` — body `{ "leaf_hash": "0x...", "digest_ts": ... }` → returns inclusion proof (list of sibling hashes) or 404.

2.9.3 Audit cross-check protocol

- Auditors fetch digests from random m validators; compare `merkle_root`.
- If mismatches, request inclusion proofs from both sides to trace divergence.
- Discrepancy triggers governance notification; signed digests are evidence for slashing.

2.10 Anti-Sybil & node admission

- Node keys signed into P2P ID.
- Admission: stake OR PoW token (configurable).
- Reputation based on uptime/correctness; repeated misbehavior → ban.

2.11 Testing checklist (Module2)

- gossip latency under churn, digest correctness, partition recovery tests, replay injection tests.

Module 3 — Service Delivery & Transport

3.1 Access model

- Services subscribe by `service_id` to index and attempt to decrypt `enc_contact` entries.

3.2 Connection bootstrap

Flow:

1. Client constructs hash & submits ledger entry (Module1).
2. Validators propagate; service matches & decrypts enc_contact.
3. Service gets `client_contact` (relay token, websocket conn id, etc.) and connects via relay or client-initiated socket.
4. Mutual ECDH (X25519) → session keys; use DTLS/SRTP for media or secure WebSocket/QUIC for data.

3.3 Transport options

- **Direct** (preferred): WebRTC (DataChannel), QUIC.
- **Relay-assisted**: TURN-like relays or LHDNS relays.
- **Multipath**: split on different relays for unlinkability.

3.4 Relay model & incentives

- Relays earn micro-payments via off-chain channels.
- Proof-of-forwarding required for rewards (tiny cryptographic receipts).
- Relays stake tokens for trust; low-reputation relays are deprioritized.

3.5 Failure modes & recovery

- If entry expired or session broken → client re-resolves & retries (exponential backoff).
- Under service overload, enforce nonce validation or require micro-fee.

Module 4 — Privacy & Anonymity Enhancements

4.1 Goals

Sender/receiver anonymity, unlinkability, resist traffic-analysis.

4.2 Query privacy

- Optional **onion submission**: client routes submission via multiple relays to hide origin (3 hops default).
- **Cover traffic & padding**: nodes inject dummy entries; standardize entry size.
- **Batching**: group submissions in 100ms windows.

4.3 Ledger privacy

- Index blinding: per-epoch salt for hashed identifiers to prevent long-term correlation (careful: requires epoch sync).
- Digests contain only digested leaf hashes; auditors hold proofs but not entries.

4.4 Transport privacy

- Multipath forwarding; relay rotation; optional mixnet integration for high-sensitivity flows.

Module 5 — Trust, Governance & Sybil Resistance

5.1 Numeric & adaptive defaults (bootstrap)

- POW_BASE_BITS = 16; POW_ADAPT_K = 2.0; clamp [8,28].
- STAKE_MIN_VALIDATOR = 10k LHD; STAKE_MIN_RELAY = 100 LHD.
- REPUTATION_DECAY_HALF_LIFE = 30 days.

5.2 Adaptive PoW formula

```
observed = submissions_last_minute / active_clients_estimate
target_rate = 1.0  # per-client-per-minute baseline
delta = log2(max(observed/target_rate, 1.0))
new_bits = clamp(POW_BASE_BITS + round(POW_ADAPT_K * delta), 8, 28)
```

Nodes include `new_bits` in digest so clients adapt PoW.

5.3 Reputation & slashing

- Reputation = moving-window metric of uptime, correctness, forwarding ratio.
- Misbehavior → slashing proportionate to stake.

5.4 Governance model

- On-chain param updates through proposals/votes (stake+reputation weighted).
- Emergency fast-quorum path for active attacks (time-locked adjustments).

Module 6 — Integration & Interoperability

6.1 Gateways

- Translate DNS names ↔ `service_id` descriptors.
- Provide bootstrap descriptors for first-time connections (search engines, indexes).
- Gateways are trusted proxies for migration; minimize their use for privacy.

6.2 Dual-stack & Fallback policy

- LHDNS local node exposes a stub resolver (127.0.0.1:53 or DoH endpoint).
- **No silent fallback** to DNS — client must prompt user or have explicit config to allow automatic fallback.
- Gateways must declare logging & manifest.

6.3 Enterprise opt-in logging

- Service side can request opt-in logging; enc_contact contains consent flag.
- Logging gateways must store logs off-ledger under enterprise control and publish signed receipts.

Module 7 — Security & Threat Model

7.1 Actor models

- Local adversary (ISP), regional observer, global passive adversary (GPA), global active adversary (GAA), colluding relays.

7.2 Threat→Mitigation (short)

Expand into a table for implementers (see later test & audit).

(See "Threat→Mitigation table" near end.)

Module 8 — Performance, Sharding & Scalability

8.1 Sharding / partitioning

- Partition by first byte of `service_id` (256 partitions); each partition replicated R = `REPLICATION_FACTOR`.
- Node assignment balanced by capacity score; cluster bridges for cross-region subscriptions.

8.2 Epidemic gossip scaling

- O(log N) convergence expected; fanout adaptive.

8.3 KPI targets (operational)

- Descriptor fetch: P50 <150ms / P90 <300ms / P99 <800ms.
- Entry propagation: P50 <200ms / P90 <800ms / P99 <2s.
- E2E secure bootstrap: P50 <500ms / P90 <1.2s / P99 <2.5s.
- Node memory: target <100k active entries.

Module 9 — Monitoring, Metrics & Auditing

9.1 Metrics exposure

- `/metrics` endpoint (Prometheus style): histograms for submission latency, propagation latency, digest mismatch count, peer-count, memory usage.

9.2 Audit procedures

- Auditors periodically fetch `GET /digest/latest` from a quorum, cross-check roots; request inclusion proofs as needed; report misbehavior to governance.

Module 10 — Example End-to-End: Anonymous Chat (Detailed)

(Condensed sequence including canonical message examples)

1. **Descriptor & Nonce** (service publishes descriptor):

```
{
 "service_id":"svc:sha256:af12...bc",
 "service_pubkey":"<b64(pub)>",
 "accepted_protocols":["webrtc","websocket"],
 "nonce_policy":{"rot_interval":30}
}
```

2. **Client prepares ephemeral context**

- Generate `A_eph_priv/pub`. Choose onion path [R1,R2,R3].

3. **Compute hash_token**

- `time_window=floor(now/30)`
- `hash=SHA256(LE32(len(A_eph_pub))||A_eph_pub||service_id||nonce_service||LE64(time_window))`

4. **Build enc_contact_plain (CBOR canonical)**

```
{
 "client_eph_pub":"<b64>",

"client_contact":{"type":"websocket","relay":"relay.example.net","conn_id":"xyz123"},
 "ts":169xxxxxxx,
 "nonce_client":"rand128",
 "client_sig":"<sig_on(hash||ts)>"
}
```

5. **Encrypt enc_contact (Module1) → build entry → submit via onion**
6. **DLN propagation (Module2)** → service subscription sees entry → decrypts → verify `client_sig` → use `client_contact`.
7. **Session establishment (Module3)**: service connects to relay or client-initiated socket; ECDH → session keys; end-to-end AEAD used for media/data.
8. **Teardown**: after TTL entry removed; bloom filters short-lived prevent replay.

Expected latencies (practical)

- Descriptor fetch: 100–300 ms.
- Propagation to service: 200 ms — 1s typical.
- Session bootstrap: ~200–800 ms depending on relays.

Test Vectors, Schema & Example JSON (for implementers)

Example canonical CBOR signing input (illustrative)

- CBOR-encode canonical object `{ "hash": "0x9f2a...6d", "ts": 169..., "service_id": <32 raw> }` and sign with Ed25519.

Test vector (concrete)

- Provide known inputs and expected `hash` + `sig` for implementers to verify.

(Implementers: request full test vector file if needed — can be generated deterministically from seed values.)

Threat → Mitigation Table (expanded)

Threat	Likelihood	Impact	Mitigation	Residual Risk
Sybil (fake nodes)	Medium	High	PoW + stake + reputation (Module 5)	Reduced, not zero
Traffic analysis	High	High	Onion submission, multipath, cover traffic	GPA remains possible
Descriptor poisoning	Medium	Medium	Signed descriptors, gossip validation, digest audits	Low
Replay attacks	Medium	Medium	Nonce + TTL tokens, clock skew policy	Low
DoS (node flooding)	High	High	Rate-limit, adaptive PoW, micro-fees, slashing	Manageable
Key compromise	Low	High	Ephemeral key rotation, forward secrecy (X25519 + HKDF), rapid revocation	Still critical if long-term keys leaked
Relay collusion	Medium	High	Multipath routing, stake-weighted relay selection, proof-of-forwarding	Risk if large relay set compromised
Clock desync	Medium	Medium	CLOCK_SKEW ±60s, fallback nonce_server_proof, adaptive PoW under skew	Possible false negatives under severe desync

Operational notes & implementation checklist (summary)

- Implement canonical CBOR pipeline precisely (JSON→JCS→CBOR if using JSON).
- Provide SDKs that hide canonicalization for application developers.
- Implement digest collector/auditor service from day 0 (safety).
- Phase rollout: private testnet → incentivized public testnet → pilot integrations (browsers/gateways) → mainnet.

Appendix: API endpoints (example)

- `POST /entry` — submit ledger entry (returns 202 accepted or error).
- `GET /resolve?service_id=<id>` — returns active entries for a service (for gateway/local debugging).
- `GET /digest/latest` — signed digest.
- `POST /proof` — inclusion proof request.
- `/metrics` — Prometheus metrics.

Authentication & rate-limit: local nodes may require bearer tokens for high-volume submissions (per implementer policy).

Closing / final remarks

This Annex A is the *final integrated technical report* — canonical serialization, crypto bindings, PoW/stake numbers, Merkle digest format + APIs, sharding plan, KPIs, testing checklists, privacy mitigations, and end-to-end example flows are included and linked to specific modules. Values marked as DEFAULTS are **bootstrap** values: they must be tuned on-testnet and then set via governance.