# SPEC-1B — Lucid RDP: Method, Governance & Consensus

## Method

### High‑Level Architecture (with full blockchain operations in MVP)

@startuml  
skinparam componentStyle rectangle  
skinparam shadowing false  
  
package "Raspberry Pi 5 (Ubuntu Server)" {  
 [Admin UI (.onion, Next.js/Node 20)] as AdminUI  
 [RDP Host (xrdp/Wayland)] as RDP  
 [Session Recorder] as Rec  
 [Chunker+Compressor (Zstd)] as Chunker  
 [Encryptor (XChaCha20-Poly1305)] as Enc  
 [Merkle Builder] as Merkle  
 [On-System Chain Client] as OSC  
 [DHT/CRDT Node] as DHT  
 [Tron-Node Client (TronWeb 6)] as TronClient  
 [Wallet Daemon (Ledger HW or SW Vault)] as Wallet  
 [Local DB (MongoDB 7, WiredTiger)] as DB  
 [Local Encrypted Chunk Store] as Store  
 [Tor HS & SOCKS Proxy] as Tor  
}  
  
package "On-System Data Chain (separate from TRON)" {  
 [LucidAnchors] as OSAnchors  
 [LucidChunkStore] as OSChunks  
}  
  
package "TRON (Mainnet/Testnet)" {  
 [PayoutRouterV0] as PR0  
 [PayoutRouterKYC] as PRKYC  
 [USDT-TRC20] as USDT  
}  
  
AdminUI -down-> RDP  
RDP --> Rec  
Rec --> Chunker --> Enc --> Store  
Enc --> Merkle  
Enc --> OSC : push chunks (encrypted)  
Merkle --> OSC : anchor(manifest/root)  
OSC --> OSAnchors  
OSC --> OSChunks  
AdminUI --> DB  
AdminUI --> TronClient : purchases, payouts  
Wallet -left- TronClient  
DHT <..> DHT : gossip/replicate (encrypted indexes)  
Tor .. AdminUI  
Tor .. OSC  
Tor .. TronClient  
Tor .. RDP  
PR0 --> USDT  
PRKYC --> USDT  
@enduml

**Rationale**: The appliance records a session locally, encrypts chunks, computes per‑session Merkle roots, and anchors them on‑chain via **LucidAnchors**. Real‑money payouts use USDT‑TRC20 via **PayoutRouterV0** (no KYC) or **PayoutRouterKYC** (KYC‑gated). Both routers are deployed **now** to avoid any contract migrations later; the admin chooses which one to use per deployment mode.

### On‑Chain Contracts (Solidity for TVM)

**1) LucidAnchors** (immutable after deploy) - registerSession(bytes32 sessionId, bytes32 manifestHash, uint64 startedAt, address owner, bytes32 merkleRoot, uint32 chunkCount) → emits SessionRegistered. - anchorChunk(bytes32 sessionId, uint32 index, bytes32 chunkHash) → emits ChunkAnchored (optional granular anchoring). - Gas‑efficient design: prefer **event logs** for anchors over storage writes; storage used only for minimal index to prevent duplicate roots. - Governance: ownership **renounced at deploy**, no upgrade proxy.

**2) PayoutRouterV0** (prod, no‑KYC route) - Holds USDT balance; callable by ROLE\_DISBURSER (the Pi or a designated service account) with audited off‑chain reason codes. - disburse(bytes32 sessionId, address to, uint256 amount) transfers **USDT (TRC20)** from the vault to to and emits Paid. - Circuit breakers: pausability + per‑tx max to contain risk. Parameters set at deploy; role holder in a 2‑of‑3 multisig.

**3) PayoutRouterKYC** (prod‑ready, KYC route) - Same as V0 plus ComplianceSigner check: disburseKYC(...) requires ECDSA signature over (to, kycHash, expiry) from the compliance key. - Optional dailyLimit[to] for payout caps. - Deployed **alongside** V0; the Pi selects which router to invoke based on policy.

**Token & chain** - **Payments**: **TRON Mainnet + USDT‑TRC20** for real payouts; **Shasta** for sandbox testing. All TRON interactions are handled by the isolated **Tron‑Node System** service over Tor; no other service calls TRON. - **Data**: The **On‑System Data Chain** (distinct from TRON) stores **encrypted, lossless‑compressed** session chunks and manifests on‑chain. A redundant encrypted copy remains on the user device.

### Off‑Chain Data & Proofs

* **Manifest**: JSON with sessionId (UUIDv4), participant pubkeys, codec, chunk count, root hash, recorder SW version, device fingerprint.
* **Chunking**: 8–16 MB before compression; **Zstd** level 3; each encrypted with **XChaCha20‑Poly1305** (per‑chunk nonce; key derived from session key via HKDF‑BLAKE2b).
* **Root**: Merkle over H(chunkCiphertext) using BLAKE3; store root + chunkCount in **LucidAnchors on the On‑System Data Chain**.
* **Local DB** (MongoDB 7, WiredTiger):
  + sessions collection: { \_id: UUID, owner\_addr, started\_at, ended\_at, manifest\_hash, merkle\_root, chunk\_count, anchor\_txid }
  + chunks collection: { \_id, session\_id: UUID, idx, local\_path, ciphertext\_sha256, size\_bytes }
  + payouts collection: { \_id, session\_id: UUID, to\_addr, usdt\_amount, router, reason, txid }
  + **Sharding**: chunks sharded on { session\_id: 1, idx: 1 }; sessions and payouts replicated.

### Key Management & Roles

* **Hardware wallet** (Ledger) for the admin multisig and cold vault; appliance wallet is software‑based with encrypted keystore + passphrase; optional HSM later.
* Roles: DEPLOYER (burned), DISBURSER (multisig), COMPLIANCE\_SIGNER (separate key, only for PRKYC), PAUSER (multisig).

### Sequence (Anchor & Payout)

@startuml  
skinparam shadowing false  
actor Operator as Op  
participant "Pi Admin UI" as UI  
participant "Recorder" as Rec  
participant "Merkle Builder" as MB  
participant "LucidAnchors (On-System Chain)" as LA  
participant "PayoutRouter(V0|KYC) on TRON" as PR  
participant "USDT-TRC20" as USDT  
  
Op -> UI : Start Session (mint single-use ID)  
UI -> Rec : record()  
Rec -> MB : finalize(manifest, hashes)  
MB -> LA : registerSession(..., merkleRoot, chunkCount)  
LA --> MB : txid (On-System Chain)  
Op -> UI : Request payout (policy)  
UI -> PR : disburse(sessionId, to, amount[, KYC sig])  
PR -> USDT : transfer(to, amount)  
USDT --> PR : Transfer event  
PR --> UI : Paid(txid) (TRON)  
@enduml

### S & C Items: Blockchain‑Sensitive Decisions (locked in MVP)

* **S3‑compatible backups (S)**: off‑chain only; no on‑chain changes.
* **Observer role (S)**: manifests reference optional observer pubkeys; no on‑chain change.
* **KYC & payout caps (C)**: addressed by deploying **PRKYC** now; choose router per policy without redeploys later.
* **Federation/Replication (C)**: off‑chain; unaffected on‑chain.
* **Post‑quantum crypto (C)**: testnet‑only recording encryption; on‑chain unchanged.
* **Privacy filters (C)**: off‑chain redaction; hashes still anchor final ciphertext; on‑chain unchanged.

### Libraries & Versions (arm64, Pi‑verified)

* Node.js **20 LTS**, TronWeb **^6**, **MongoDB Server 7** + Node driver **^6** (or Mongoose **^8**), Python **3.12** optional. Tron resource model (Energy/Bandwidth) informs staking/rental to minimize fees.

## Governance Addendum (MVP — Immutable From Launch)

**Principle:** The MVP ships with **full on-chain governance** and an **immutable block system**. **No upgrades or edits** to the chain logic or contract bytecode are permitted **after launch**, including during the test phase.

### Governance Scope (On‑System Data Chain)

* **LucidGovernor + Timelock** (EVM‑style on the On‑System Data Chain): proposal → vote → timelock → execution. Contracts are **non‑upgradeable**; governance can only adjust **explicit parameters** in a **ParamRegistry** and perform **role rotations** where allowed.
* **ParamRegistry (immutable interface):** bounded, typed keys (e.g., payoutEpochLength, maxChunkSize, policyTimeoutMs). Only the Governor may set values within pre‑defined safe ranges. No arbitrary code paths.
* **Roles & Voting Power:**
  + **Server (Original node)** — genesis role; bootstrap validator, initial proposer rights.
  + **Node Workers** — validators/replicators; **one‑node‑one‑vote** among **eligible** nodes (≥80% uptime over prior 3‑month window **and** ≥1 LUCID earned). Tokens **do not** amplify governance votes.
  + **Admin** — operational multisig; executes Governor decisions (where off‑chain action is needed).
  + **Dev** — runs a node worker; proposal rights during MVP to unblock fixes to configs (not code), subject to vote.
* **Immutability Controls:**
  + All application contracts (**LucidAnchors, LucidChunkStore, ParamRegistry, Governor**) are **final**, **ownership renounced**, **no proxies**.
  + Chain consensus parameters (gas limit, block time, validator set size formula) are frozen in genesis or changed **only** via the Governor if and only if the genesis rules allow such parameter changes. No code upgrades.

### Cross‑Chain Governance (TRON Payouts)

* **Authority Selection:** On‑System Governor controls the **list of TRON signer keys** (2‑of‑3 multisig) for payout actions. Rotation is effected by an on‑chain vote that updates a **SignerRegistry** (On‑System), mirrored to the **Tron‑Node System**.
* **Routers Deployed Now:** **PRKYC** (for node‑worker payouts) and **PR0** (for non‑worker/end‑user flows) are both deployed at MVP and remain unchanged post‑launch.
* **Monthly Epochs:** Governance selects the **epoch Merkle root submitter** and parameters (e.g., claim window). The TRON distributor honors only calls signed by the active multisig.

## Centralised Relay Point (“Lucid Cloud Relay”)

**Definition:** A Tor‑reachable, onion‑addressed relay layer composed of **the same actors** that operate the network: - **[Server] Original node** — bootstrap relay and directory. - **[Node workers]** — contribute relay bandwidth and directory shards while also performing storage/validation. - **[Admin]** — runs node worker(s) with additional ops dashboards and throttling controls. - **[Dev]** — runs node worker(s) used for test/sandbox while still participating in relay duties.

**Functions:** - Rendezvous/directory (.onion) for session introduction and DHT bootstrap (still P2P data paths afterward). - Rate‑limited, audited relay for metadata when direct peer connectivity is poor (still over Tor; no clearnet). - **MongoDB** replica/shard routers (mongos) exposed via .onion; workers host shards/replicas.

**Constraints:** - Relay runs **only over Tor**; it is logically central but **physically decentralized** across the above roles. - Participation counts toward **monthly work credits** and thus voting power.

## Consensus Addendum — Proof of Operational Tasks (PoOT)

**Goal:** The **block publisher is selected by a tally of operational tasks**, not by hash power. The **node or node‑pool with the highest verified operational work** in the recent window claims the right to publish, with deterministic fallbacks for liveness.

### Inputs to the Tally (per 1‑hour slots, rolled up monthly)

* **Relay bandwidth** served via Lucid Cloud Relay (.onion), signed metering beacons.
* **Storage availability** proofs for encrypted chunk replicas.
* **Validation signatures** on On‑System Chain blocks/transactions.
* **Uptime beacons** (time‑sealed heartbeats).

All task proofs are **Tor‑routed**, signed, and written to MongoDB (task\_proofs) and periodically committed on‑chain by **WorkCreditsOracle**.

### Leader Selection (per block slot)

1. Compute WorkCredits over a **sliding window** (e.g., last 7 days) within the current epoch.
2. Rank **entities** (node or **node‑pool**). Pools are identified by a **poolId** and backed by a multisig.
3. **Primary leader = top‑ranked entity** not selected in the past cooldownSlots.
4. **Fallbacks:** if the primary does not propose within slotTimeoutMs, the next ranked entity claims the slot, etc. A missed slot penalizes the entity’s liveScore for the remainder of the epoch (prevents griefing).
5. **Tie‑break:** use a **VRF** seeded from the previous block hash to pick the winner among entities with equal credits.

@startuml  
skinparam shadowing false  
participant Scheduler  
participant WorkCreditsOracle as WCO  
participant "Entity A (pool)" as A  
participant "Entity B" as B  
  
WCO -> Scheduler : credits(slidingWindow)  
Scheduler -> Scheduler : rank entities, apply cooldown  
Scheduler -> A : grant slot (deadline = slotTimeoutMs)  
A -> Scheduler : propose(block) or timeout  
alt A proposes  
 Scheduler -> All : accept block  
else timeout  
 Scheduler -> B : grant slot  
end  
@enduml

### Pools

* Multiple nodes can form a **pool**; their work proofs aggregate to the poolId.
* The pool publishes with a **multisig** key and handles **internal revenue split** off‑chain (not protocol‑enforced).

### Separation from Governance

* **Consensus leader selection uses WorkCredits.**
* **Governance voting remains one‑node‑one‑vote** (eligibility: ≥80% uptime over last 3 months and ≥1 LUCID earned).

### MongoDB Collections (consensus)

* task\_proofs: { \_id, nodeId, poolId?, slot, type, value, sig, ts } (sharded on { slot: 1, nodeId: 1 }).
* work\_tally: { \_id: epoch, entityId, credits, liveScore, rank } (replicated).
* leader\_schedule: { \_id, slot, primary: entityId, fallbacks: [entityId...], result: { winner, reason } } (replicated).

## Consensus Parameters (Set for MVP)

* **slotDurationSec = 120s** → **max 30 blocks/hour** (immutable).
* **slotTimeoutMs = 5000** (leader has 5s to propose before fallback inside the slot).
* **cooldownSlots = 16** (leader cannot be scheduled again for at least 16 slots).
* **leaderWindowDays = 7** (sliding window for PoOT tally).

**ParamRegistry keys (bounded)** - slotDurationSec (fixed **120**) - slotTimeoutMs (default **5000**, bounds **[1000, 10000]**) - cooldownSlots (default **16**, bounds **[8, 64]**) - leaderWindowDays (default **7**, bounds **[3, 14]**)

## Method — Metrics & Collections (updates)

* session\_events: { \_id, sessionId, type: "started|ended", ts\_slot, nodeId, sig } (replicated; used to compute S\_t).
* slot\_metrics: { \_id: slot, sessions\_total: S\_t, reward\_raw, reward\_scaled, publisher, txid } (replicated).