# Orogen

Verifiable LLM inference, anchored to physical work.

Whitepaper v0.1 — May 2026
OROG · orogen.network
github.com/orogen-network/llm-mining

# Contents

# Part I — Network architecture and rationale

# Plan — Full Blockchain Implementation for an LLM Mining Network

## Context

The user has spent a year producing a research dossier in `ref/` (10 sections — proposal, proposal-v1, landscape, review, verifiability, training, economics, gpu-infra, bittensor, pearl). The target architecture is committed: Substrate fork + EVM JSON-RPC, 7 custom pallets, 8-layer verification stack, worker daemon on vLLM V1, gateway/router, validator network, content-addressed weight CDN, attestation chain.

Two downstream artifacts are committed ( `projects/ecosystem-evolution` , `projects/libertai-pearl-partnership` ). Repository is otherwise clean.

This plan is the *implementation* plan — how to build the chain end-to-end (chain, node, mining client, inference protocol, gateway, validator, explorer, wallet, on-ramp), bake in every red-team mitigation, and execute autonomously on a multi-year horizon with `/goal` -driven feedback loops. It is calibrated to the proposal's anchors: ~$15–20M budget, ~30 FTE peak, Q2 2027 TGE.

If any anchor shifts (budget, schedule, GTM assumptions), the gates in §10 trigger a re-plan rather than a slip.

---

## 0. /goal Autonomous Execution Layer

This is the meta-layer. The plan itself is structured so an autonomous agent can drive it forward on a long horizon with a single top-level `/goal` and recurring status pulls. **Never delegate understanding** — the agent's job is to advance the plan and surface decisions, not to redefine the plan.

## 0.1 Goal hierarchy

```
G0: Ship a bulletproof LLM mining chain to mainnet by Q2 2027, hold subsidy <2× Y1
└─ G1: $1M ARR pre-TGE (Q4 2026)              [gate: 12.2]
├─ G2: Pallet-suite v0.9 feature-complete    (Q4 2026)
├─ G3: Public testnet "Forge" stable 6 mo    (Q1 2027)
├─ G4: Multi-firm audit clean                (Q2 2027)
├─ G5: Mainnet TGE                           (Q2 2027)   [gate: 12.3]
├─ G6: Permissionless transition             (Q3 2027)   [gate: 12.4]
└─ G7: Subsidy <1× by 2028                   [gate: roadmap KPI]
```

Each Gn has explicit success criteria (§10 gates), a primary owner, and a feedback signal (§0.3).

## 0.2 Execution model

- **Plan file is durable** at `/home/jon/.claude/plans/make-a-full-plan-groovy-gosling.md`. Updates are diffs.
- **Status file** at `STATUS.md` (repo root, gitignored or committed weekly) — autonomously maintained: current Gn, gate state, blockers, last KPI snapshot, next decision needed.
- **Inbox file** at `DECISIONS.md` — open decisions awaiting human input, one per H2 heading, with options + recommended default + deadline. Agent never auto-decides items here.
- `/goal "advance the LLM mining plan"` reads the plan + STATUS + DECISIONS, picks the next blocking action, executes/researches, updates STATUS, and surfaces any new DECISIONS.

## 0.3 Feedback loop

Each goal has one **leading signal** (catches slip early) and one **lagging signal** (proves it shipped):

| Goal | Leading signal | Lagging signal | Cadence |
|---|---|---|---|
| G1 ($1M ARR) | Pilot pipeline value × close-rate | Booked ARR | Weekly |
| G2 (pallet-suite) | LoC merged into `pallet-suite` + fuzz coverage % | Integration-test E2E green for 7d | Weekly |
| G3 (Forge stable) | Operators online (7d heartbeat) + chaos pass rate | 6 mo continuous operation, no P0 unresolved | Daily |
| G4 (audit) | Findings open by severity | All P0/P1 closed + re-verified | Weekly during audit |
| G5 (TGE) | All §12.3 boxes ticked | Mainnet block production stable 30 d | Daily T-30 → T+30 |
| G6 (permissionless) | Day-30 burn:mint ratio | Permissionless operator T-to-first-job <2h | Daily |
| G7 (subsidy <1×) | Monthly subsidy ratio | Rolling 90-d subsidy <1× sustained | Monthly |

Signals are scraped from: GitHub (LoC, CI), gateway DB (ARR), chain RPC (operators, blocks, burns, mints), Immunefi (bounty reports). Auto-aggregated into the public `subsidy-dashboard`.

### 0.4 Scheduled automation

- **Daily 09:00 UTC:** signal scrape → STATUS update → diff posted to `#mining-status` (private). If any leading signal trips a threshold, opens a DECISIONS entry.
- **Weekly Mon 14:00 UTC:** gate review across all Gn. If any G's gate slips its forecast by >2 weeks, opens a re-plan DECISIONS entry.
- **Quarterly (first Mon of quarter):** full milestone review, KPI dashboard update, public progress post.
- **Continuous:** chaos harness nightly results → STATUS; bounty triage SLA-tracked.

Recommended setup: `/schedule` for the daily/weekly/quarterly cadence using a single agent that runs the signal-scrape + STATUS update routine. `/loop 1h /chaos-watch` during the 4-week shadowfork window. Manual `/goal` invocations between scheduled passes for human-directed acceleration.

### 0.5 Autonomous re-planning rules

The agent may **propose** but never **execute** a re-plan. Triggers: - Any §10 gate criterion misses by >20% with <30 days runway → propose mitigation or slip. - Any red-line in §13 is at risk → halt, escalate. - Any RFC contract (§7) breaks compatibility test → freeze branch, propose contract revision. - Budget burn >110% of plan at any quarter → propose scope cut.

All re-plans land in DECISIONS.md with options, recommended default, deadline.

---

## 1. Repo / Codebase Layout

Single GitHub org. ~36 repos total. Public/private flag per repo. Trunk-based + stacked PRs. Mixed languages — Rust for chain & cryptographic code, Python for serving/gateway, TS for frontend/SDKs, Go for CDN, C++ for edge.

## 1.1 Chain & runtime (Rust, Substrate)

| Repo | Purpose | Forks/wraps |
|---|---|---|
| `chain-node` | Node binary (Polkadot SDK), Frontier EVM, GRANDPA, AURA→BABE later | polkadot-sdk, frontier |
| `pallet-suite` | Cargo workspace with `pallet-{model-registry,operator-stake,job-market,yuma-consensus,bme,slashing,pouw-mint,attestation-registry,oracle-twap,nonce-vault,treasury-ext}` | subtensor (reference only) |
| `runtime-mainnet` / `runtime-testnet` | Runtime composition, spec versions, storage migrations | — |
| `chain-tooling-rust` | subxt-based CLIs: extrinsic submitter, chain-spec generator, genesis builder, slash-receipt verifier, RFC/spec docs | subxt |
| `bridge-snowbridge-fork` | Eth bridge (post-TGE, not launch-blocking) | Snowbridge |
| `chain-indexer` | Subsquid archival indexer for explorer + dashboards | subsquid |

Four pallets *added* beyond the original 7: `attestation-registry` (TEE quotes + CRL), `oracle-twap`, `nonce-vault` (customer nonces), `treasury-ext`. They are non-negotiable given red-team rules 1, 10, 11, 14.

## 1.2 Verification & attestation

| Repo | Purpose |
|---|---|
| `validator-replay` | Stake-weighted sampler (10% floor, commit-reveal randomness, Yuma scoring), TOPLOC vendored, SGLang det-mode for replay |
| `validator-watcher` | **Separate binary, pool-disjoint** (red-team rule 5): log-prob drift, BME burn integrity, anomaly detection |
| `attestation-service` | Aggregates NVTrust RIM + Intel TA + AMD KDS into one signed report; multi-vendor mandatory at launch (rule 8) |
| `opml-challenger` | opML challenge window (high-value jobs only, Phase 4) |
| `zkml-prover` | EZKL primary; Polyhedra/Lagrange fallbacks. Small heads only (moderation, routing). |
| `cuPOW-pool` | Optional 5% emission lane (Hopper-only, deferred to Q4 2028) |

## 1.3 Serving (operator-side)

| Repo | Purpose | Lang |
|---|---|---|
| `infer-worker-vllm` | Primary daemon. Pinned vLLM V1 fork, plugin shim, watchdog, single Docker image, auto-update (rule 13) | Python |
| `infer-worker-sglang` | SGLang variant (chat/RAG/deterministic) | Python |
| `infer-worker-trtllm` | TensorRT-LLM (dc-premium only) | Python/ C++ |
| `infer-worker-llamacpp` | Edge / embed-only | C++ |
| `worker-control-plane` | Heartbeat, capability advertisement, on-chain registration, stake-bind, log shipping. Shared across worker variants | Rust |
| `weight-cdn-pinner` | S3 + Cloudflare R2 + IPFS, hash-verify-on-pull | Go |
| `adapter-registry-svc` | Off-chain LoRA blob server + on-chain metadata via `pallet-model-registry` | Rust |

## 1.4 Gateway / customer-facing

| Repo | Purpose | Lang |
|---|---|---|
| `gateway-router` | OpenAI-compatible API, nonce mgmt, batch settlement, drives BME burn | Python |
| `litellm-backend-driver` | LiteLLM provider plugin — acquisition channel | Python |
| `customer-sdk-py` / `customer-sdk-ts` | OpenAI-shaped SDKs | Python/ TS |
| `billing-bridge` | Pre-TGE Stripe/Coinbase Commerce/Transak USD ingestion → CUC (post-TGE: optional on-ramp) | TS |
| `gateway-burn-engine` | Separated burn execution (auditable). Watches gateway revenue, executes BME extrinsics, publishes burn-integrity metrics. **Required separate from gateway** — rule 1 | Rust |

## 1.5 Wallets

| Repo | Purpose | Lang |
|---|---|---|
| `wallet-sdk-core` | Shared key derivation (sr25519 + secp256k1), signing, RPC | Rust + WASM |
| `wallet-cli` | First wallet (operator/validator onboarding) | Rust |
| `wallet-extension` | Browser ext (SS58 + EVM in one UI) | TS |
| `wallet-mobile` | iOS+Android — **deferred to Q4 2027** | RN |

## 1.6 Explorer / dashboards

| Repo | Purpose |
|---|---|
| `explorer-web` | Blocks/extrinsics/accounts/models/operators/slashes; fork polkadot-js-apps |
| `subsidy-dashboard` | Public subsidy ratio, burn:mint, validator concentration, geo distribution (KPIs from §0.3) |
| `attestation-explorer` | Verify a signed receipt + replay the TEE chain + show operator reputation |
| `status-page` | Health, RPC endpoints, oracle freshness, sanctions feed freshness |

## 1.7 Ops / compliance / DevEx

| Repo | Purpose |
|---|---|
| `operator-onboarding-ui` | 1-click web: GPU detect → TEE quote → stake-bind → wallet gen → daemon install |
| `testnet-faucet` | Faucet + rate limiter + sybil resistance |
| `terraform-foundation` | IaC: foundation RPCs, sentinel validators, telemetry, weight CDN |
| `chaos-harness` | Adversarial operator/validator/oracle simulator. Nightly against testnet (rule 15) |
| `incident-runbooks` | Markdown runbooks for §6 scenarios |
| `sanctions-screener` | Chainalysis + TRM + Elliptic; continuous; pushes block list to `pallet-operator-stake` (rule 16) |
| `governance-tools` | Multisig admin, dispute panel UI, slash appeals, CRL ops |
| `docs-site` | docs.useful.network (Docusaurus) |

---

# 2. Critical path & sequencing

## 2.1 The longest path

**Pallet-suite v0.9 → runtime composition → public testnet stability → 6-month soak → audit cycle → mainnet.**

~14 months wall-clock under perfect execution. Q3 2026 pallet-suite start → Q4 2026 v0.9 → Q1 2027 testnet stable + audit → Q2 2027 audit-fix → TGE end Q2 2027. **Zero slack.**

Compression levers (already baked in): 1. Senior chain hires onboard by W0 (T−12 months). 2. Two-firm audits run in parallel on different scopes (Trail of Bits on pallets, OpenZeppelin on bridge/EVM, both on economic — rule 15). 3. Operator + gateway stack on permissioned dev chain by Q4 2026 so chain integration is the only late integration. 4. **Cross-team contracts (§7) ratified by end Q3 2026** — single biggest leverage point.

## 2.2 Build waves

```
 W0 (Q2 2026, T−12 mo): founders + first 5 (CTO, Chain Lead, Head of GTM, GC, Foundation
Ops)
W1 (Q3 2026, T−9 mo):  chain core, verification lead, serving lead, CISO, DevOps lead
W2 (Q4 2026, T−6 mo):  pallet seniors, gateway, wallet, frontend, sec engineer, SE,
pilot PM
W3 (Q1 2027, T−3 mo):  audit liaison, mid-level fillers, second SE
W4 (Q3 2027, T+3 mo):  RL/research team for Phase 2 (post-TGE only)
```

### 2.3 Phase-by-phase

| Phase | Quarter | Deliverable | Customer | KPI |
|---|---|---|---|---|
| 0 Stealth | Q3 2026 | Centralized stack + 3 paying pilots + chain skeleton on devnet | 3 design partners | 3 pilots, $30K+ MRR, >99% gateway uptime |
| 1 Pre-TGE revenue | Q4 2026 | $1M ARR; pallet-suite v0.9; worker daemon → devnet; wallet CLI; multi-vendor attestation green on CI | 5+ logos | $1M ARR, pallet integration test 7-d green |
| 2 Forge + audit | Q1 2027 | Public testnet open; chaos harness nightly; two-firm audit kicked off; sanctions screener live; onboarding UI ships | External operators | ≥100 operators registered, ≥50 sustained 7-d heartbeat |
| 3 TGE | Q2 2027 | Audit clean; mainnet genesis (16–24 validators); gateway migration; 3 independent RPCs; Day-30 KPI gate | Token holders, customers, operators | All §10.3 gate criteria green |
| 4 Permissionless + opML + LoRA | Q3 2027 | Permissionless onboarding (if Day-30 green); opML for high-value; LoRA marketplace; foundation share <50% | Open community | ≥10 adapters, ≥1k jobs/30d, <20% single-entity stake |
| 5 Phase 2 prep | Q4 2027 | Mobile wallet; prime-rl + verifiers + Atropos + OpenRLHF vendored; subsidy <1.2× tracking | Retail wallet users; research | 10k+ MAU mobile, first lab RL run |
| 6 Phase 2 RL | H1 2028 | First customer RL pilot; zkML moderation/routing heads | 1–2 RL customers | Event-free RL run |
| 7 Phase 3 federated | H2 2028 | SparseLoCo federated post-training; cuPOW-pool optional lane | Research customers | Subsidy <1× holding |

---

## 3. Headcount & team

Peak ~34 FTE. Fully-loaded ~$300K/FTE → ~$20M over 24 months. Hire order (first 10): CTO → Chain Lead → Head of GTM → Verification Lead → Serving Lead → CISO → DevOps Lead → Pallet Senior #1 → Pallet Senior #2 → Solutions Engineer.

| Function | Peak FTE | Wave |
|---|---|---|
| Chain / runtime / pallets | 5 | W0 lead, W1 rest |
| Verification (replay, watcher, attestation, opML/zkML) | 4 | W1 lead, W2 rest |
| Serving infra (vLLM/SGLang/TRT-LLM workers) | 4 | W1 lead, W2 rest |
| Gateway / router / burn engine | 3 | W2 |
| Wallet / SDKs / DevEx | 3 | W2 |
| Explorer / dashboards / frontend | 2 | W2 |
| DevOps / SRE / chaos | 3 | W1, W2 |
| Security / audit / sanctions | 2 | W1 CISO, W2 second |
| GTM / pilot / CS | 3 | W0 head, W1 SE, W2 PM |
| Founders / exec | 3 | W0 |
| Foundation ops (legal, comms, corp-dev) | 2 | W0, W2 |
| **Total peak** | **34** | |

## 4. Cross-team integration contracts — ratify by end Q3 2026

Locked at RFC level before parallel build. Versioned. Breaking changes require all-leads sign-off + CI compat-test bump.

1. **Signed response receipt** — `(job_id, operator_id, model_id, model_weight_hash, customer_nonce, request_hash, response_hash, log_probs_sample, kv_metadata, kernel_pack_hash, gpu_model, driver_version, attestation_report_hash, timestamp, operator_signature)` . SCALE on-chain, JSON off-chain. ed25519.
2. **Attestation report ABI** — combined Intel TDX + AMD SEV-SNP + NVIDIA CC quotes + RIM. ≥1 for permissionless, ≥2 for dc-premium, all 3 for compliance tier. Hash on-chain via `pallet-attestation-registry` .
3. **Heartbeat schema** — `(operator_id, capabilities[], current_load, kv_cache_pressure, last_completed_job_id, attestation_freshness, watchdog_state, version, signature)` every 12s, block-aligned.
4. **Batch settlement format** — N receipts → 1 extrinsic, Merkle root, operator-signed; mints + burns in one batch (rule 1).
5. **Slashing extrinsic ABI** — `(operator_id, fault_code, evidence_hash, severity, dispute_window_end, validator_signatures[])` ; per-detection (rule 3); severity buckets 0.5/2/5/10% single-incident.
6. **Adapter registration** — `(adapter_id, base_model_id, adapter_hash, cdn_urls[], registration_fee_paid, storage_rent_until, creator_id, metadata_uri, signature)` . On-chain stores hash + URLs only (rule 9).
7. **Customer nonce protocol** — customer `(nonce, ts, sig)` per inference; operator rejects duplicate within 24h (rule 11); `pallet-nonce-vault` records hashes.

8. **Oracle feed** — TWAP 4–12h (extended for 18 mo post-TGE per rule 10); ≥4 sources within 5%; static fallback.
9. **Operator registration** — `(coldkey, hotkey, device_cert, geo_region, ip_/24_hash, tier, stake_amount, attestation_quote, sanctions_check_proof, signature)`. Same `device_cert` across coldkeys → both slashed 100% (rule 7).
10. **RPC endpoint contract** — 3 independent providers commit to standard Substrate JSON-RPC + Frontier EVM JSON-RPC + WS subscriptions + archive availability + uptime SLA on public status page.

RFCs land in `chain-tooling-rust/specs/`. Compat-test suite in CI on every contract change.

---

## 5. OSS strategy

| Dependency | Strategy | Upstream contributions |
|---|---|---|
| polkadot-sdk | Vendor + light fork, 1-version lag | Bugfixes only |
| Frontier | Vendor, pinned | Gas-metering tweaks |
| subtensor pallets | **Reference only**, do not inherit | None |
| Snowbridge | Fork (will diverge); not launch-blocking | Post-TGE |
| **vLLM** | Vendor pinned + plugin shim | **Yes — push our hook spec upstream (moat)** |
| **SGLang** | Vendor, track det-mode work | **Yes — kernel determinism fixes** |
| TensorRT-LLM / Dynamo | Integrate only (NVIDIA-managed) | Bug reports |
| llama.cpp / MLX | Integrate | Bug reports |
| **LiteLLM** | Integrate + ship `litellm-backend-driver` upstream | **Yes — official provider plugin (acquisition channel)** |
| vLLM production-stack / llm-d | Pick one by end Q4 2026 | Operator-side patches |
| NVTrust / Intel TA / AMD KDS | Remote services via SDKs | Bug reports |
| **TOPLOC** | Vendor + extend | **Yes — nonce + attestation hash fields** |
| EZKL / Polyhedra / Lagrange | Integrate (default EZKL), keep adapters for others | Issue feedback |
| prime-rl / verifiers / Atropos / OpenRLHF | Vendor + integrate (Phase 2) | Post-launch |
| SparseLoCo | Vendor (Phase 3) | Maybe |
| polkadot.js extension | Fork | None |
| polkadot-js-apps (explorer) | Fork | Bugfix PRs |
| subxt / subsquid | Integrate | Bugfixes |

Strategic upstream targets — dedicated eng time: (1) vLLM hooks, (2) SGLang det-mode, (3) LiteLLM provider, (4) TOPLOC extensions.

---

## 6. Verification stack — design rules baked in

8-layer per the proposal, with red-team mitigations:

- **L1 GPU device cert + stake binding** — collision → both coldkeys slashed 100% (rule 7); hardware-rooted, not stake-only.
- **L2 TEE** — Intel TDX + AMD SEV-SNP + NVIDIA H100/H200/B200 CC. Multi-vendor at launch (rule 8). On-chain CRL with 48-h foundation SLA + 7-d operator grace (§8).
- **L3 Deterministic kernels** — SGLang det-mode + Thinking Machines batch-invariant kernels. ~30–40% throughput cost priced into protocol; non-deterministic = cheaper tier marked best-effort.
- **L4 Stake-weighted validator sampling** — Yuma-style. **Sample rate ≥10% floor** (not 1–5%), commit-reveal randomness with 1-epoch delay (rule 2). Per-detection slashing (rule 3). Top-K=128 validator set, outlier-clipped.
- **L5 opML challenge window** — high-value jobs only (Phase 4, not TGE-blocking). 1h/24h/7d windows. Watchers bonded ≥$10K. Pool-disjoint from validators (rule 5).
- **L6 zkML** — small heads only (moderation, routing, classification). Two-prover-system for moderation; not on emission-critical paths. Independent verifier impls required.
- **L7 cuPOW** — minting only (5% supply, optional, Hopper). Activation-commitment chain to bind matmul→model. Activate via governance Q4 2028 if economics warrant.
- **L8 Watermarks** — provenance only, not correctness. Christ-Gunn-Zamir undetectable + publicly-detectable variant.

Per-tier verification matrix unchanged from proposal ( `dc-premium` , `dc-standard` , `cloud-rented` , `prosumer` , `edge` , `embed-only` ).

---

# 7. Testnet & audit plan (bulletproofing)

## 7.1 Five named networks

| Net | When | Participants | Lifetime |
|---|---|---|---|
| `mining-devnet` | Q3 2026+ | Foundation eng | Perpetual (always 1 runtime ahead) |
| `mining-public-testnet` (Forge) | Q4 2026 launch | Partners then public, ~30–50 operators | ≥6 months (cannot end before TGE) |
| `mining-incentivized-testnet` (Forge-Stake) | Q1 2027 | ~200 KYC'd operators | ≥3 months overlap with Forge |
| `mining-security-testnet` (Forge-Adversarial) | Q1 2027+ | Red teams, bounty hunters | Perpetual |
| `mining-shadowfork` | Mid Q2 2027 | All planned mainnet validators + 50% operators | 4 weeks |

Forge-Stake rewards: **0.5% of supply** pre-allocated, pro-rata to honest mining + bug reports + chaos participation. Vested 6-mo cliff + 18-mo linear. Capped at $50K notional/operator (anti-sybil).

## 7.2 Audit cadence

Multi-firm cross-audit on every critical pallet (rule from Wormhole/Ronin/Nomad history):

| Component | Primary | Cross-audit |
|---|---|---|
| `pallet-yuma-consensus` | Trail of Bits | Sigma Prime |
| `pallet-bme` | OpenZeppelin | Zellic |
| `pallet-slashing` | Trail of Bits | Sec3 |
| `pallet-pouw-mint` | Sigma Prime | OpenZeppelin |
| Attestation verifier | NCC Group | Trail of Bits |
| Oracle code | Zellic | ConsenSys Diligence |
| `pallet-job-market` | Quantstamp | — |
| `pallet-operator-stake` | Halborn | — |
| `pallet-model-registry` | Halborn | — |
| EVM JSON-RPC / bridge | ConsenSys Diligence | — |
| Worker daemon | NCC Group | — |
| Gateway | Halborn | — |
| Validator client | Sigma Prime | — |
| Multisig + timelock | OpenZeppelin | Trail of Bits |
| Tokenomics math | OpenZeppelin | Gauntlet (actuarial) |

Budget: ~$1.85M pre-TGE. Anything less is corner-cutting.

Re-audit triggers: pallet diff >100 LOC since last audit, any change to critical math, new pallet, new host-function, vendor PKI root change, oracle source change, upstream CVE, peer-protocol exploit of shared class.

## 7.3 Continuous security testing

- **Fuzzing** — `cargo-fuzz` + `honggfuzz` on pallets; `echidna` + `foundry` on EVM surfaces. 85% line / 70% branch on critical pallets. Per-pallet targets defined.
- **Property-based** ( `proptest` ) — stake conservation, emission cap, burn-mint ratio, slash math non-negativity, sampling-fairness uniformity, no-double-spend, timelock honored, vesting honored.
- **Formal verification** — Kani/Creusot on `pouw-mint::compute_reward` , `bme::check_cap` , `slashing::compute_slash` . Bounded-int symbolic execution. Anything not provably bounded → $10^6$ proptest iter.
- **Chaos harness** — 13 scenarios nightly: lazy validator, sybil farm, oracle manipulation, validator-operator collusion, network partition, RPC outage, BAR0 side-channel, firmware downgrade, quantization swap, prompt-caching exploit, model substitution, slashing-runaway, mass-unstake.
- **Red-team engagements** — 4-week Trail of Bits red team Q1 2027; 2-week Spearbit/Zellic offensive Q2 2027 focused on Q1 findings.

### 7.4 Bug bounty (Immunefi + direct PGP channel)

Live Day 1 of Forge, never paused.

| Severity | Definition | Payout |
|---|---|---|
| Critical | Chain halt, mint-from-nothing, total fund theft, attestation bypass all vendors, governance override | up to $500K |
| High | Slashing exploit, single-vendor attestation bypass, BME accounting error, partial fund theft, validator collusion | up to $100K |
| Medium | DoS without funds at risk, oracle manipulation under specific conditions | up to $25K |
| Low | Info leak, minor griefing, incorrect event | up to $5K |

90-day disclosure embargo; coordinated disclosure with affected operators T−7d before public. Annual pool post-TGE: $2M.

---

## 8. Release management & incident response

### 8.1 Runtime upgrade pipeline (no exceptions)

1. Merge to `main` behind feature flag → 2. Devnet 7-d soak → 3. Forge/Forge-Stake 14-d → 4. Shadowfork 7-d + rollback drill → 5. Re-audit if triggered → 6. On-chain governance proposal → 7. **14-day timelock** → 8. **2-day public delay** even for emergency 5-of-7 fast-track → 9. `pallet-system::set_code` → 10. 48-h elevated on-call → 11. Postmortem 7 d.

Storage migrations versioned + inverse defined + tested with mainnet-state snapshot in shadowfork. Last 3 runtime wasms maintained for rollback.

### 8.2 Incident playbooks (10 scenarios)

Each scenario in `incident-runbooks/` : detection signal, owner, immediate commands, escalation chain, comms template, postmortem requirement.

1. Chain halt (block age >300s) → validator-ops lead, 15-min escalation, 60-min public statement.
2. Slashing storm (>10× baseline/h) → security lead, `mining-cli slashing pause` via multisig, emergency upgrade if bug.
3. Oracle stuck/manipulated (>15% deviation, >30 min stale) → oracle ops, failover to backup, BME pause if needed.
4. Validator collusion detected (cross-validator score >3σ outlier) → security lead, forensic 24h, arbitration panel.
5. TEE CVE disclosed (vendor PSIRT/SA feed) → attestation lead, CRL entry within 48h, 7-d operator grace (§9).
6. Exchange listing manipulation → foundation ops + outside counsel, clarifying statement, **no foundation trading**.

7. Large unlock dump → no intervention by design; 180-d rolling cap absorbs.
8. RPC DDoS → infra lead, Cloudflare WAF, failover; chain validators on separate network.
9. Fake-burn / customer fraud (burn velocity >5σ) → compliance lead, freeze via CRL, void mints via multisig.
10. Regulatory subpoena → general counsel, statutory ack, litigation hold, outside counsel.

On-call: 4 engineers, 1-week rotation, PagerDuty + Opsgenie. Security council reachable in 15 min for P0.

### 8.3 Slashing dispute & arbitration

- Slashed stake **escrowed not burned** until T0+28d.
- T0+7d: operator opens dispute, posts 10% dispute bond.
- T0+14d: on-chain sortition selects 3-operator panel from top-50 stake (≠ slashed, ≠ slashing validator); each posts 1% bond.
- T0+21d: panel votes 2-of-3.
- T0+28d: 5-of-7 council ratifies or overrides; resolution executed.
- False dispute → bond burned. Bad-faith dispute → additional 25% slash. False watcher claim → bond×10 + ban on 2nd offense.

### 8.4 TEE CRL flow

CRL entries (revoked device certs, firmware hashes, model hashes, sanctioned wallets) live in `pallet-attestation-registry`, IPFS mirror. Operator daemon checks every 10 min + on every job start + on attestation refresh. After CRL update: T0+7d grace → T0+14d soft-slash 5% → T0+30d full deregistration.

### 8.5 Sanctions screening

Triple-vendor: Chainalysis (primary) + TRM (secondary) + Elliptic (tertiary). Screen at registration + every 24h continuously + every 1h on inbound transfers + real-time on withdrawal. Kill-switch via 3-of-7 multisig fast-track (sanctions are non-controversial), stake frozen pending OFAC resolution. **Operators and validators only — not retail customers on permissionless chain.**

---

## 9. Foundation legal & regulatory

### 9.1 Two-entity split (Howey defense)

- **MiningLabs Inc.** — Delaware C-corp. Runs SaaS gateway pre-TGE and the compliance-tier gateway post-TGE. Books $1M ARR. Customer contracts. Employees.
- **Mining Foundation** — Swiss Stiftung (Zug) or Cayman Foundation Co. Holds chain governance, multisig, treasury, grants. No for-profit motive. No pre-TGE revenue.

Hard separation: different banks, different governance, different team rotation. Pre-TGE customers can stay on USDC/fiat; not required to hold token.

### 9.2 Multisig

5-of-7 signers, geo + org distributed: - 2× Foundation council - 2× C-corp executives - 1× Independent technical advisor - 1× Independent legal/compliance advisor - 1× Community-elected (post-permissionless; held by independent steward until Q3 2027)

Hardware wallets, key ceremony, quarterly rotation drill, 1-year cooldown on outgoing signer beyond vested tokens.

### 9.3 Jurisdictional gates

| Jurisdiction | Strategy |
|---|---|
| US | Geo-fence retail at TGE; accredited only (Reg D 506(c) / Reg S); SaaS revenue (Inc.) is unaffected |
| EU | MiCA white paper filed Q1 2027; BaFin/AMF engagement |
| UK | Treat as restricted communication; HNW + sophisticated only (Travers Smith / Fieldfisher) |
| Singapore | MAS PSA structuring; engage Rajah & Tann |
| Switzerland | FINMA pre-ruling on classification |

EU AI Act: foundation is distributor, not provider; adapter authors are providers; operator-level deployer obligations in ToS; Article 50 watermarking default-on for EU traffic.

**HIPAA** / **PCI:** default tier is NOT compliant — marketed explicitly. Separate compliance tier with BAA + audit + SOC 2 / HITRUST-certified operators. Do not claim HIPAA without BAA.

---

## 10. Go/no-go gates

Each gate has explicit criteria. If not green, transition does not happen. We communicate publicly why.

### 10.1 Devnet → Public testnet (Forge)

- All 7 pallets compile under stable Rust
- E2E integration test 7 d
- ≥3 distinct vendor TEEs validated
- Fuzz coverage ≥70% line on critical pallets
- Devnet 30 d no manual restart
- Bug bounty live on Immunefi

### 10.2 Forge → Forge-Stake

- ≥30 operators attested across ≥2 vendors
- ≥10 independent validators

- 90 d continuous Forge operation
- No P0 bugs >7 d open
- BME validated >1000 burns/mints, ratio held
- First audit slate (multisig + tokenomics + 2 critical pallets) clean

### 10.3 Forge-Stake → Mainnet TGE

- Full audit slate complete, all P0/P1 closed and re-verified
- Forge run ≥6 months
- Forge-Stake ≥3 months
- Shadowfork ≥4 weeks + 2 successful upgrade/rollback drills
- **$1M ARR achieved pre-TGE** (utility demonstrated for Howey)
- ≥50 operators KYC'd + screened
- No P0/P1 open from bounty or chaos
- 1 full live red-team complete, no unfixed Critical findings
- IR runbook drilled ≥4× (4 quarters)
- Legal opinions on file for US/EU/UK/SG
- Multisig signers identified, hardware enrolled, key ceremony complete
- Insurance reserve ≥5% of treasury funded
- 180-d rolling burn cap validated under simulated bear-market
- Public roadmap for permissionless transition published

### 10.4 Mainnet → Permissionless (Q3 2027)

- Day-30 burn:mint within design band
- Day-30 cumulative slash rate <5% of stake
- Day-30 dispute panel has resolved ≥1 real dispute
- Sanctions pipeline running w/o false-positive incidents
- Multisig has executed ≥1 governance action via timelock
- Self-service onboarding tested (no handholding)
- CRL exercised ≥1× (test or real)
- Community council seat filled

---

## 11. Risks & mitigations

1. **vLLM plugin instability** — pinned fork + plugin shim + push hooks upstream; SGLang fallback. Freeze on fork if two minor releases break us.
2. **Multi-vendor TEE rollout** — gate TGE on TDX + NVIDIA CC; SEV-SNP beta on testnet OK. Vendor-pluggable `attestation-service`.
3. **Pallet economic correctness** — agent-based simulation harness Day 1; two-firm overlapping audit; BME bytecode-hash committed; 5-of-7 multisig + 2-d delay for emergency fix.
4. **opML implementability** — high-value jobs only; Phase 4 not TGE-blocking; fall back to 15% L4 sampling.

5. **Substrate-EVM bridge maturity** — not on TGE path; launch with EVM JSON-RPC via Frontier only; Snowbridge ships Q1 2028.
6. **Determinism across GPU SKUs** — per-tier float epsilon; receipts include exact GPU/driver/CUDA/kernel-pack hash; validators replay on matched hardware.
7. **Weight CDN poison/DoS** — triple-source (S3 + R2 + IPFS), hash verify, hot models foundation-pinned.
8. **Validator concentration creep** — hardware-rooted identity bound to device cert; same device cert → both slashed 100%; geo + IP-diversity required.
9. **AMD MI3xx production readiness** — AMD = `dc-standard` only at launch; not in `dc-premium`; not marketed as parity.
10. **Sanctions/regulatory whiplash** — continuous screening; permissioned compliance tier as separate operator class; Inc./Foundation split gives jurisdictional optionality.
11. **Howey defense** — GC + outside crypto counsel from W0; foundation legally separate; pre-TGE revenue to Inc. only; 4y cliff + 4y linear unchanged.
12. **Pilot retention through audit cycle** — Solutions Eng + Pilot PM hired Q3/Q4 2026 to hold pilots; no customer migration until post-mainnet.
13. **Chain-lead bus factor** — 2 Senior pallet engineers hired immediately after lead; architectural docs maintained in `pallet-suite/docs`.
14. **Determinism throughput penalty** — deterministic mode mandatory for `dc-premium`; customers pay for verifiability tier.

---

## 12. Out of scope (through TGE / first 12 months)

Hard NOs through Q2 2027: - Full transformer zkML (small heads only) - Cross-operator KV sharing - Mixed-vendor tensor parallel - Federated pretraining-from-scratch (Phase 3 territory) - Subnet AMMs / sub-tokens (rule: one network, one token) - Passive staking yield - Mobile wallet at TGE (Q4 2027) - Exotic LBP/fair-launch tokenomics (CEX-tier-1 + Uniswap v3 only at TGE) - DAO governance for protocol params at TGE (5-of-7 multisig for first 12 months) - Customer-side training jobs (inference only at TGE) - AMD `dc-premium` (Phase 4 earliest) - Bridges beyond Ethereum - Per-token usage-based vesting unlocks

---

## 13. Red lines — never do these

Each is a discipline lesson. Breaking any requires public memo + council vote.

1. No fiat-direct backdoor bypassing BME (Render lesson)
2. No foundation buyback of token (multiple)
3. No marketing token as investment / no APY language (Howey/SEC)
4. No claiming HIPAA/PCI without formal BAA/audit (io.net)
5. No fast-tracking audits to hit launch date (Wormhole/Ronin)
6. No skipping 6-month public testnet (multiple rugs)
7. No emission/slashing/BME governance override without 14-d timelock + 2-d public delay — even mid-exploit

8. No node-license-as-debt model (Aethir)
9. No 100×+ subsidy gap between emission and revenue (Bittensor)
10. No pre-listing limbo without real revenue (Pearl)
11. No silence on GPU spoofing (io.net's 12-month silence)
12. No single-firm audit on critical pallets
13. No anonymous council seats during permissioned phase
14. No private side-deals with operators or validators
15. No SAFT secondary trading pre-TGE
16. No discretionary slashing
17. No undisclosed token holdings by council/foundation
18. No emergency upgrade without 2-day public delay

---

## 14. Open architecture decisions (recommended defaults)

Captured here so the `/goal` agent knows what's defaulted vs needs human input. To override, post to `DECISIONS.md`.

| Decision | Default |
|---|---|
| Wallet format | Both SS58 + secp256k1 sub-account derived for EVM (Bittensor/Moonbeam pattern); UI shows both |
| Frontier-EVM L1 vs L2 rollup | Frontier at L1 for TGE; reassess L2 at Phase 3 |
| Validator/operator binary | Split — different lifecycles; some hosts install both |
| Mobile wallet at TGE | Defer to Q4 2027 |
| RPC nodes | Foundation runs 1 + partners run 2 = 3 at launch (rule 12) |
| Token symbol | `USFL` (4-char exchange ticker for `OROG`) |
| CUC credit | On-chain, non-transferable, account-soulbound |
| Burn-engine key | 2-of-3 FROST threshold with HSM custody |
| Sample rate | 10% floor, variable per tier (edge 25%, dc-premium 10%), commit-reveal 1-epoch delay |
| Receipt storage | Off-chain blob + Merkle root on-chain; indexer + IPFS for cold archive |
| Finality | GRANDPA at launch; BEEFY when bridge ships |
| Block production | AURA at launch (16–24 foundation validators); BABE at permissionless |
| Pre-TGE points | Yes — Forge-Stake operator rewards, capped 0.5% supply, vested 6mo cliff + 18mo linear |
| Permissioned compliance tier | Same chain, flag in `pallet-operator-stake`; gateway routes flagged operators only |
| TGE distribution | Hybrid: CEX-led (Coinbase/Kraken/Binance) for liquidity + small Coinlist KYC sale; **no public US retail** |
| Subsidy ratio measurement | USD-based 30-d rolling internal; both published externally |
| `pouw-mint` lane | Deferred. 0% genesis allocation. Activate Q4 2028 via governance if economics warrant. |

---

## 15. Critical files to create / modify (entry points)

- `/home/jon/repos/llm-mining/STATUS.md` — autonomously maintained status (§0)
- `/home/jon/repos/llm-mining/DECISIONS.md` — open decisions inbox (§0)
- `/home/jon/repos/llm-mining/chain-tooling-rust/specs/RFC-0001-receipt-format.md` … `RFC-0010-rpc-contract.md` — the 10 cross-team contracts (§4)
- `/home/jon/repos/llm-mining/pallet-suite/runtime/src/lib.rs` — runtime composition + pallet wiring + timelock config
- `/home/jon/repos/llm-mining/pallet-suite/pallets/bme/src/lib.rs` — burn-mint math + 180-d rolling cap (formal-verification target)

- `/home/jon/repos/llm-mining/pallet-suite/pallets/slashing/src/lib.rs` — slash math + dispute hooks + escrow-not-burn (FV target)
- `/home/jon/repos/llm-mining/pallet-suite/pallets/yuma-consensus/src/lib.rs` — validator scoring + outlier clipping + collusion signals (multi-firm audit)
- `/home/jon/repos/llm-mining/services/attestation-verifier/` — multi-vendor PKI + CRL consumption (multi-firm audit)
- `/home/jon/repos/llm-mining/incident-runbooks/` — markdown runbooks for §8.2 scenarios
- `/home/jon/repos/llm-mining/chaos-harness/` — nightly adversarial simulator (§7.3)

---

## 16. Verification (how we know it works end-to-end)

Three levels of end-to-end test:

1. **Devnet integration test** — `pallet-suite` workspace test that runs the full lifecycle in a single test: register model → register operator → stake → place job → operator runs vLLM stub → signs receipt → validator replays → submits Yuma weight → epoch boundary → mint emitted → operator paid → CUC consumed → state reconciles. Runs in CI on every PR. Must pass for 7 consecutive nightly runs before a Forge cut.

2. **Forge end-to-end real-hardware test** — real H100 + Intel TDX, real attestation against NVIDIA RIM, real burn-mint flow with `tFORGE` token, real validator replay across ≥3 independent validators. Tracked as `chaos-harness` daily heartbeat. 90 days continuous = gate 10.2.

3. **Shadowfork mainnet rehearsal** — full runtime upgrade + rollback drill against mainnet-config state. Two upgrade/rollback drills successful = gate 10.3. One IR drill executed end-to-end = gate 10.3.

For UI components (explorer, dashboards, operator-onboarding-ui, wallets): launched and exercised against Forge. Each UI must demonstrate the golden path (operator onboarding <15 min p50, customer first-job <60 s, explorer query latency <500 ms) in front of the team before claiming "done."

For autonomous execution itself: `/goal "advance the LLM mining plan"` should produce a coherent STATUS update and at most one new DECISIONS entry per invocation, with no fabricated signals. Drift-detect by running it weekly and reviewing the STATUS diff. If it ever auto-decides a §14 default without DECISIONS approval, halt the autonomous loop and re-tune.

---

## TL;DR

- **One immovable date:** Q2 2027 TGE. Everything back-plans from there.
- **One long pole:** pallet-suite v0.9 by end Q4 2026.
- **One biggest leverage point:** cross-team contracts (§4) ratified by end Q3 2026 so the rest can parallelize.
- **One discipline:** if any §10 gate isn't green, we don't ship. Period.

- **One feedback loop:** `/goal` reads plan + STATUS + DECISIONS, advances, updates, surfaces. Never decides §13 red lines or §14 defaults autonomously.

Bulletproof is a discipline, not a slogan. This plan is the discipline.

# Part II — Cross-team integration RFCs

# RFC-0001 — Signed Response Receipt Format

**Status:** Draft — ratification target end Q3 2026 **Owner:** Verification Lead **Consumers:** `validator-replay`, `validator-watcher`, `gateway-burn-engine`, `pallet-job-market`, `customer-sdk-{py,ts}`, `attestation-explorer` **Cross-cuts:** RFC-0002 (attestation report), RFC-0004 (batch settlement), RFC-0005 (slashing extrinsic), RFC-0007 (nonce protocol)

## Goal

Define the canonical signed response receipt emitted by an operator after each inference. Receipts are the unit of work measurement, dispute, and BME settlement.

## Fields

```
 #[derive(Encode, Decode, Clone, PartialEq, Eq, RuntimeDebug, TypeInfo)]
pub struct Receipt {
    pub version: u8,                    // 1
    pub job_id: H256,                   // unique per inference
    pub operator_id: AccountId,         // operator hotkey
    pub model_id: H256,                 // base model content hash from pallet-model-
registry
    pub model_weight_hash: H256,        // exact weight tensor hash served (catches
quant swap)
    pub adapter_id: Option<H256>,       // LoRA adapter, if any
    pub customer_nonce: H256,           // RFC-0007 nonce
    pub request_hash: H256,             // SHA-256 of canonical-serialized request
    pub response_hash: H256,            // SHA-256 of canonical-serialized response
    pub log_probs_sample: Vec<u8>,      // first 64 token log-prob distributions
(Targon-style)
    pub kv_metadata: KvMetadata,        // prefix hint, cache state, see below
    pub kernel_pack_hash: H256,         // serving-engine + kernel version pinning
    pub gpu_model: BoundedString<32>,   // "H100-SXM-80GB"
    pub driver_version: BoundedString<32>,
    pub cuda_version: BoundedString<32>,
    pub attestation_report_hash: H256,  // points to pallet-attestation-registry entry
(RFC-0002)
    pub batch_invariant_proof: Option<H256>, // SGLang det-mode proof, if applicable
    pub timestamp_ms: u64,
    pub gateway_id: AccountId,          // who routed this job
    pub operator_signature: Signature,  // ed25519 over
BLAKE2-256(canonical_encode(everything above))
}

#[derive(Encode, Decode, Clone, PartialEq, Eq, RuntimeDebug, TypeInfo)]
pub struct KvMetadata {
    pub prefix_hint: Option<H256>,    // first 1024 token hash for KV-aware routing
    pub cache_hit: bool,              // did we hit KV cache
    pub kv_blocks_used: u32,          // for capacity tracking
}
```

## Encodings

- **On-chain:** SCALE codec. Stored only as Merkle root in batch settlement (RFC-0004). Individual receipts off-chain.
- **Off-chain blob:** JSON with the same field names, base64 for hashes and signatures. Stored in `chain-indexer` archival + IPFS pin for receipt cold-archive.
- **Wire (customer SDK → gateway):** JSON.
- **Wire (operator → gateway):** JSON for HTTP, SCALE for gRPC.

## Signature

```
sig = ed25519_sign(operator_hotkey, BLAKE2-256(SCALE-
encode(receipt_without_signature_field)))
```

Verification path: `validator-replay` reads receipt, derives canonical bytes, verifies `operator_signature` against `operator_id` hotkey registered in `pallet-operator-stake`.

## Replay protocol

A validator that samples this receipt (RFC-0006 random selection):

1. Pull receipt blob from `chain-indexer` or operator's gateway-pinned URL.
2. Re-fetch `(model, adapter)` from `pallet-model-registry` content-addressed URLs.
3. Replay using `(request_hash, model_weight_hash, kernel_pack_hash, gpu_model, driver_version, cuda_version)` to match operator's environment within per-tier float ε.
4. Re-compute `response_hash` and `log_probs_sample`.
5. Compare to receipt. Mismatch → slashing extrinsic (RFC-0005).

## Limits

- `log_probs_sample`: capped at 64 tokens × 16-bit indices × top-5 probs = 640 bytes. Sufficient for Targon-style check; bounded for chain cost.
- Total receipt size: ~1.5 KiB worst case.

## Errors / dispute

If operator signature is invalid → receipt is dropped, no slash (not chain-verifiable that operator was even involved). If signature valid but content mismatches replay → slashing extrinsic with severity per-fault-code:

| Fault code | Severity | Example |
|---|---|---|
| `WrongModel` | 10% | model_weight_hash ≠ replayed |
| `WrongResponse` | 5% | response_hash ≠ replayed |
| `LogProbDrift` | 2% | log_probs diverge beyond ε |
| `CacheReplay` | 5% | cache_hit=true but no fresh-compute signal |
| `KernelPackMismatch` | 0.5% | non-deterministic kernel used in deterministic tier |

## Versioning

`version: u8 = 1` at TGE. Increment on any breaking field change; gateway and worker daemon must accept current + previous version for one runtime cycle.

## Open questions

- Should `log_probs_sample` be operator-chosen 64 tokens or randomly-chosen (commit-reveal) for stronger adversary model? Default: random via per-receipt seed = `BLAKE2-256(customer_nonce || job_id)`.
- Should `request_hash` cover raw text or canonicalized chat-template-applied form? Default: canonicalized; chat-template version pinned in `kernel_pack_hash`.

## Backward compatibility

None at TGE. Post-TGE, additive fields only via `version` bump + tail-padding rules.

# RFC-0002 — Combined Multi-Vendor Attestation Report

**Status:** Draft — ratification target end Q3 2026 **Owner:** Security Lead **Consumers:** `attestation-service`, `pallet-attestation-registry`, `validator-replay`, `operator-onboarding-ui`

## Goal

Define how Intel TDX + AMD SEV-SNP + NVIDIA H100/H200/B200 CC attestation quotes are combined into one signed report, hashed, and stored on-chain.

## Why multi-vendor

Red-team rule 8: single-vendor PKI compromise should not collapse the network. A combined report binds the operator to *all* attesting vendors simultaneously; a verifier rejects if any required vendor's quote is missing or invalid.

## Vendor matrix per tier

| Tier | Required quotes | Optional quotes |
|---|---|---|
| `dc-premium` | NVIDIA CC + (Intel TDX OR AMD SEV-SNP) | second CPU vendor |
| `dc-standard` | NVIDIA CC + (Intel TDX OR AMD SEV-SNP) | — |
| `cloud-rented` | NVIDIA CC + Intel TDX | — |
| `prosumer` / `edge` / `embed-only` | none (stake-only sybil resistance) | — |
| `compliance` (HIPAA/PCI tier) | NVIDIA CC + Intel TDX + AMD SEV-SNP | SOC 2 cert hash |

## Report structure

```
 #[derive(Encode, Decode, Clone, PartialEq, Eq, RuntimeDebug, TypeInfo)]
pub struct AttestationReport {
    pub version: u8,                  // 1
    pub operator_id: AccountId,
    pub tier: OperatorTier,
    pub gpu_quote: Option<NvidiaQuote>,
    pub tdx_quote: Option<IntelTdxQuote>,
    pub sev_snp_report: Option<AmdSevSnpReport>,
    pub rim_attestation: Option<NvtrustRimAttestation>,
    pub firmware_hashes: BoundedVec<H256, ConstU32<16>>,  // for CRL check
    pub measured_vm_bundle: H256,     // what's running inside the enclave
    pub timestamp_ms: u64,
    pub validity_window_ms: u64,      // re-attest required after this elapses
    pub aggregator_signature: Signature, // signed by attestation-service
    pub vendor_pki_chain_hashes: BoundedVec<H256, ConstU32<8>>, // for CRL lookup
}

#[derive(Encode, Decode, Clone, PartialEq, Eq, RuntimeDebug, TypeInfo)]
pub struct NvidiaQuote {
    pub device_cert: BoundedVec<u8, ConstU32<8192>>,
    pub attestation_cert: BoundedVec<u8, ConstU32<8192>>,
    pub measurement: H256,
    pub nonce: H256,                  // anti-replay
    pub gpu_uuid: H256,               // silicon-bound identity
}

// IntelTdxQuote and AmdSevSnpReport follow vendor formats; canonicalized hashes stored
on-chain
```

## Storage

`pallet-attestation-registry` stores only:

```
 pub struct OnChainAttestation {
    pub operator_id: AccountId,
    pub report_hash: H256,            // = BLAKE2-256(SCALE-
encode(report_without_signature))
    pub gpu_uuid: H256,               // for L1 device-cert collision detection (red-team
rule 7)
    pub vendor_set: BitFlags8,        // which vendors are in this report
    pub measured_vm_bundle: H256,
    pub expires_at: BlockNumber,
    pub revoked: bool,
}
```

Full report blob lives in `chain-indexer` + IPFS. `attestation-service` produces report, computes hash, calls `pallet-attestation-registry::submit(report_hash, gpu_uuid, vendor_set, measured_vm_bundle, expires_at)` via aggregator-signed extrinsic.

## CRL (Certificate Revocation List)

```
pub struct CrlEntry {
    pub kind: CrlKind,    // FirmwareHash | DeviceCert | ModelHash | VendorPkiChain
    pub target: H256,
    pub reason: BoundedVec<u8, ConstU32<256>>,
    pub added_at: BlockNumber,
    pub grace_until: BlockNumber,
}
```

CRL writes are multisig-gated (5-of-7) with the standard 14-day timelock — *except* for sanctions hits (3-of-7 fast-track) and known-CVE firmware hashes (3-of-7 fast-track per IR playbook §8.2 #5).

Operators check CRL via `pallet-attestation-registry::is_revoked(report_hash | gpu_uuid | firmware_hash)` every 10 minutes and on every job start.

## Re-attestation

- Every 7 days minimum (validity_window default = 7 × 86400 × 1000 ms).
- Immediately after any CRL update affecting this operator.
- Triggers a re-submit; old report_hash is `revoked=true` but receipts older than the new attestation are honored.

## Vendor PKI chain validation (off-chain)

`attestation-service` validates against:

- **NVIDIA NVTrust** — RIM service, Device Identity CA, Attestation CA. Chain pulls latest CA bundle from a foundation-hosted pinned mirror; updates via governance.
- **Intel Trust Authority** — quote signing CA, fmspc-bound endorsements.
- **AMD KDS** — root-of-trust certificate, ASK certificate.

A vendor's PKI chain compromise → governance push to CRL all `vendor_pki_chain_hashes` matching the compromised root.

## Side-channel disclosure

Some side channels exist that this attestation does not mitigate: - Hopper unencrypted NVLink (acknowledged; route confidential workloads to Blackwell B200/B300 when available). - BAR0 register leakage (arxiv 2507.02770). - Bimodal timing channels (batch-size leakage). - GPUBreach (shared-timing).

These are listed in operator ToS and customer-facing docs. Network does not claim protection against silicon-undisclosed channels.

## Versioning

`version: u8 = 1` at TGE. Increment on any vendor matrix change. Backward-compat by tail-padding only.

## Open questions

- Whether to gate `dc-premium` on Blackwell-only operators once the Hopper NVLink issue is publicly catalogued. Default: market-discovered; expose `nvlink_encryption: bool` to customer routing filter.

# RFC-0003 — Heartbeat Schema

**Status:** Draft — ratification target end Q3 2026 **Owner:** Serving Lead **Consumers:** `gateway-router` (for routing), `validator-watcher` (anomaly detection), `pallet-operator-stake` (liveness)

## Goal

Define the recurring liveness + capability advertisement an operator emits while online. Heartbeats are *not* receipts; they're the lightweight feed that lets the gateway pick capable operators and the chain detect downtime.

## Cadence

- **Off-chain heartbeat** (gateway-bound): every 12 seconds, block-aligned. Pushed over WebSocket from `worker-control-plane` to gateway router fleet.
- **On-chain heartbeat** (liveness anchor): once per epoch (every 360 blocks ≈ 72 minutes). Extrinsic to `pallet-operator-stake::heartbeat()`. Inclusion proves the operator is alive in that epoch; absence triggers liveness penalty (not slashing, just emission share reduction).

## Off-chain heartbeat structure

```
 #[derive(Serialize, Deserialize, Clone)]
pub struct OffChainHeartbeat {
    pub version: u8,                            // 1
    pub operator_id: AccountId,                 // hotkey
    pub block_number: u64,                      // current chain tip the operator saw
    pub capabilities: Vec<Capability>,
    pub current_load: LoadSnapshot,
    pub kv_cache_pressure: f32,                 // 0.0–1.0
    pub last_completed_job_id: Option<H256>,
    pub attestation_freshness: AttestationFreshness,
    pub watchdog_state: WatchdogState,
    pub price_per_million_tokens: u64,          // in CUC micro-units
    pub geo_region: BoundedString<8>,           // ISO-3166-1 alpha-2 + subdiv
    pub signature: Signature,                   // ed25519 over canonical-encode
}

#[derive(Serialize, Deserialize, Clone)]
pub struct Capability {
    pub base_model_id: H256,
    pub adapter_ids: Vec<H256>,                 // currently warm adapters
    pub quantization: Quantization,             // FP16 | FP8 | INT8 | INT4
    pub max_context_tokens: u32,
    pub max_concurrent_requests: u32,
    pub deterministic_mode: bool,
}

#[derive(Serialize, Deserialize, Clone)]
pub struct LoadSnapshot {
    pub active_requests: u32,
    pub queue_depth: u32,
    pub p50_ttft_ms: u32,
    pub p99_ttft_ms: u32,
    pub p50_itl_ms: u32,
    pub p99_itl_ms: u32,
    pub gpu_memory_used_gb: f32,
    pub gpu_utilization_pct: f32,
}

#[derive(Serialize, Deserialize, Clone)]
pub struct AttestationFreshness {
    pub last_attested_at_ms: u64,
    pub expires_at_ms: u64,
    pub current_report_hash: H256,
}

#[derive(Serialize, Deserialize, Clone)]
pub struct WatchdogState {
    pub vllm_pid_alive: bool,
    pub vllm_last_log_ms: u64,
    pub last_restart_count_24h: u32,
}
```

## On-chain heartbeat extrinsic

```
fn heartbeat(
    origin: OriginFor<T>,
    epoch_number: u64,
    capabilities_summary_hash: H256,    // BLAKE2 of canonical capabilities list
    load_summary: LoadSummary,
    attestation_report_hash: H256,      // current attestation from pallet-attestation-
registry
) -> DispatchResult;
```

`LoadSummary` is bounded; `capabilities_summary_hash` is enough to detect drift; full capabilities advertised off-chain (more frequent + bigger).

## Routing semantics

Gateway router maintains an in-memory operator catalog refreshed by off-chain heartbeats. Routing decisions per request:

1. Resolve `(base_model_id, adapter_id?, tier_preference, region_preference, latency_budget, max_price)`.
2. Filter `Capability` set: operators that advertise the requested model + adapter + tier within budget.
3. Score by latency estimate, KV-warm hint, reputation (from yuma scoring), price.
4. Session pinning: if `session_id` is provided and the prior operator is still capable, prefer them.
5. Send job; on operator timeout (>2× p99 TTFT), retry on next-best.

## Validator watcher signals

`validator-watcher` ingests heartbeats and flags: - Sudden capability churn (operator drops 10+ adapters in 1 hour without re-attestation). - Load anomaly (p99 TTFT spikes 5× while utilization stays flat — synthetic-batch attack signal). - Geo-region change without re-attestation (sybil signal). - Attestation freshness vs. CRL state (operator running stale firmware).

These are *signals*, not slashes; signals feed validator scoring and IR playbook §8.2 #11.

## Bandwidth

12s heartbeat × ~10 KiB payload × ~1000 operators × 100 gateway replicas worst-case = ~84 GiB/day total catalog traffic. Acceptable for gateway-side ingest; aggregated via per-region pub/sub.

## Versioning

`version: u8 = 1` at TGE. Increment on schema change. Mixed-version operators tolerated for one runtime cycle.

## Open questions

- Should off-chain heartbeats be signed by hotkey or by a separate session key for performance? Default: hotkey (operational simplicity), revisit at >5k operators.
- Should we gossip heartbeats over libp2p to other operators (for peer discovery) or keep gateway-only? Default: gateway-only at TGE; libp2p in Phase 4.

# RFC-0004 — Batch Settlement Format

**Status:** Draft — ratification target end Q3 2026 **Owner:** Pallet Lead **Consumers:** `pallet-job-market`, `pallet-bme`, `gateway-burn-engine`, `validator-replay`

## Goal

Aggregate ~thousands of per-inference receipts (RFC-0001) into one on-chain settlement extrinsic per epoch per gateway. Reduces 100M tx/day → ~280K tx/day total across the network.

## Settlement cadence

- One batch per epoch (~72 min) per gateway. Multi-gateway operators are encouraged for fault tolerance.
- Receipts older than 2 epochs at submission time are rejected (anti-stale).
- Receipts younger than 1 epoch are accepted (allows gateways to settle eagerly when batch is full).

## Batch structure

```
 #[derive(Encode, Decode, Clone, PartialEq, Eq, RuntimeDebug, TypeInfo)]
pub struct SettlementBatch {
    pub version: u8,                              // 1
    pub batch_id: H256,                           // BLAKE2 of all leaves + epoch +
gateway
    pub epoch_number: u64,
    pub gateway_id: AccountId,
    pub receipt_count: u32,
    pub merkle_root: H256,                        // root over leaf =
BLAKE2(canonical_encode(receipt))
    pub aggregate_burn_cuc: u128,                 // total CUC consumed in this batch
    pub aggregate_mint_useful: u128,              // expected operator mint
    pub per_operator_summary: BoundedVec<OperatorSummary, ConstU32<512>>,
    pub gateway_signature: Signature,             // ed25519
}

#[derive(Encode, Decode, Clone, PartialEq, Eq, RuntimeDebug, TypeInfo)]
pub struct OperatorSummary {
    pub operator_id: AccountId,
    pub receipts_count: u32,
    pub aggregate_tokens_served: u64,
    pub aggregate_mint_useful: u128,
    pub merkle_subroot: H256,                     // subroot over just this operator's
receipts
}
```

## Extrinsic

```
fn submit_batch(
    origin: OriginFor<T>,                          // must be a registered gateway
    batch: SettlementBatch,
    receipts_blob_cdn_url: BoundedString<256>,     // where full receipts are pinned
) -> DispatchResult;
```

Effects:

1. Verify `gateway_signature` against registered gateway hotkey.
2. Verify per-operator aggregates ≤ on-chain stake-pool capacity (anti-overcommit).
3. Verify aggregate mint ≤ epoch mint headroom from `pallet-bme`.
4. Verify aggregate burn ≥ aggregate mint × USD-equivalent ratio (per BME math).
5. Burn CUC from gateway's escrow.
6. Mint OROG to operators per `OperatorSummary`.
7. Emit `SettlementBatchSubmitted` event with `(batch_id, gateway_id, merkle_root, operator_summaries_hash)`.
8. Store batch header (not full content) in `BatchHeaders` storage.

## Dispute window

After submission:

- T+0 to T+24h: any operator can submit `dispute_batch(batch_id, merkle_proof, expected_summary)` if their `OperatorSummary` is missing or wrong. Disputed batches partially refunded; mint reversed via `pallet-bme::reverse_mint`.
- T+0 to T+epoch_end: validators replay sampled receipts (RFC-0006) and submit slashing extrinsics (RFC-0005) for mismatches.

## Receipt blob hosting

`receipts_blob_cdn_url` points to: - `chain-indexer` archival (operated by foundation + partners). - IPFS pin via `weight-cdn-pinner` (content-addressed by `batch_id`). - Gateway's own S3/R2 (mandatory until indexer is decentralized).

Operator and validator must be able to fetch any receipt via Merkle proof against `merkle_root`.

## Sanity checks (chain-side)

- `receipt_count == per_operator_summary.map(|s| s.receipts_count).sum()`.
- `aggregate_burn_cuc == receipts.map(|r| burn_for(r)).sum()` — verified by gateway-burn-engine before signing.
- `aggregate_mint_useful ≤ aggregate_burn_cuc × oracle_rate × subsidy_factor` from `pallet-bme` epoch state.

## Adversary scenarios

1. **Inflated mint claim:** gateway claims more mint than burn supports. Caught at extrinsic validation (step 4). Slash gateway 10% of stake; void batch.
2. **Fake receipts:** gateway includes receipts that operators never signed. Operators dispute via T+24h window; gateway slashed.
3. **Missing receipts:** gateway omits operators' work to save burn. Operators dispute via T+24h window; mint added retroactively + gateway slashed.
4. **Batch collision:** two gateways claim same job_id. Earliest valid wins; later slashed.

## Storage cost

Header per batch: ~10 KiB SCALE. At 280K batches/day = 2.8 GB/day on archive nodes. Pruning policy: keep all headers; full receipts off-chain via CDN.

## Versioning

`version: u8 = 1` at TGE. Increment on field changes. Compat-tested in CI.

## Open questions

- Should batch size be capped (e.g. 10K receipts max)? Default: yes, 10K cap per batch; gateway emits multiple batches if needed.
- Should we support cross-gateway batch aggregation (one batch from multiple gateways)? Default: no at TGE; complicates dispute attribution.

# RFC-0005 — Slashing Extrinsic ABI

**Status:** Draft — ratification target end Q3 2026 **Owner:** Pallet Lead + Verification Lead **Consumers:** `pallet-slashing`, `pallet-operator-stake`, `validator-replay`, `validator-watcher`, `governance-tools`

## Goal

Define how slashing evidence is submitted, how severity is calculated per-fault, how the dispute window operates, and how slashed stake is escrowed (not burned) until resolution.

## Design rules (from red-team)

1. **Per-detection, not per-epoch.** 100 cheats detected = 100 slash events. (Rule 3.)
2. **Bounded per-incident.** Max 10% single-incident slash.
3. **Cumulative cap.** Max 50% per month per operator.
4. **Dispute window.** 7 days for operator to dispute; 28 days total resolution.
5. **Escrow, not burn.** Slashed stake held in escrow until T+28d.
6. **Transparency.** Every slash on-chain with reason code; appeal mechanism.
7. **Watcher false-positive penalty.** False slashing claim → watcher bond × 10 + 2nd-offense ban.

## Fault codes

```
#[derive(Encode, Decode, Clone, PartialEq, Eq, RuntimeDebug, TypeInfo)]
pub enum FaultCode {
    WrongModel,             // 10% — model_weight_hash mismatch on replay
    WrongResponse,          // 5%  — response_hash mismatch on replay
    LogProbDrift,           // 2%  — log_probs diverge beyond per-tier ε
    CacheReplay,            // 5%  — cache_hit without fresh-compute signal
    QuantizationSwap,       // 10% — actual quant ≠ declared
    KernelPackMismatch,     // 0.5% — non-det kernel in det tier
    DeviceCertCollision,    // 100% — same GPU UUID across coldkeys (rule 7) — special,
no dispute
    HeartbeatMiss,          // soft — no slash, just emission decay
    AttestationStale,       // 2%  — operating past CRL grace period
    SanctionsHit,           // 100% — frozen, not burned (rule 16)
    ValidatorCollusion,     // 10% — cross-validator outlier confirmed
    FakeBurn,               // 50% — gateway-side fraud
    BatchOvercommit,        // 10% — gateway claimed mint > burn supports
}

impl FaultCode {
    pub fn base_severity_bps(&self) -> u16 {
        match self {
            WrongModel | QuantizationSwap | ValidatorCollusion | BatchOvercommit =>
1000,
            WrongResponse | CacheReplay => 500,
            LogProbDrift | AttestationStale => 200,
            KernelPackMismatch => 50,
            DeviceCertCollision | SanctionsHit => 10000,  // 100%
            FakeBurn => 5000,
            HeartbeatMiss => 0,  // soft
        }
    }
}
```

## Submission extrinsic

```
fn submit_slashing_evidence(
    origin: OriginFor<T>,                       // validator or watcher (pool-disjoint)
    operator_id: AccountId,
    fault_code: FaultCode,
    evidence_hash: H256,                        // points to off-chain evidence blob
    related_job_id: Option<H256>,               // for receipt-based faults
    related_receipt_hash: Option<H256>,
    validator_signatures: BoundedVec<(AccountId, Signature), ConstU32<8>>,
                                                // co-signatures for high-severity
claims
) -> DispatchResult;
```

## Multi-signature requirement by severity

| Severity | Co-signature count required |
|---|---|
| 0.5% (KernelPackMismatch) | 1 (submitter only) |
| 2–5% | 2 (submitter + 1 corroborator) |
| 10% | 3 corroborators |
| 50% (FakeBurn) | 3 corroborators + gateway-burn-engine signed evidence |
| 100% (DeviceCertCollision, SanctionsHit) | 5 corroborators OR multisig override |

Corroborators must be: - For validator-class faults (WrongModel, LogProbDrift, etc.): top-K validator set, geographically distinct. - For gateway-class faults (FakeBurn, BatchOvercommit): different gateways or independent burn-engine instances. - For sanctions: foundation multisig fast-track (3-of-7).

## Slash flow

1. Evidence submitted → extrinsic validated → `Slashing` event emitted.
2. Operator's stake equal to `severity_bps × stake / 10000` moved to **escrow account**, not burned.
3. Operator notified via off-chain event subscription.
4. Operator has 7 days to file `dispute_slashing(slash_id, dispute_bond_amount, counter_evidence_hash)` (RFC-0005 §dispute below).
5. If no dispute, at T+7d slash moves from escrow to **slashing-result account** (still not burned, waits 21 more days).
6. At T+28d, slash burns from slashing-result account, unless dispute resolved in operator's favor.

## Dispute protocol

See plan §8.3. Implementation summary:

```
fn dispute_slashing(
    origin: OriginFor<T>,                       // must be the slashed operator
    slash_id: u64,
    dispute_bond: BalanceOf<T>,                 // 10% of slash amount
    counter_evidence_hash: H256,
) -> DispatchResult;

fn arbitrate_dispute(
    origin: OriginFor<T>,                       // must be sortition-selected panelist
    slash_id: u64,
    vote: ArbitrationVote,                      // Uphold | Overturn | Insufficient
    rationale_hash: H256,
) -> DispatchResult;

fn ratify_dispute(
    origin: OriginFor<T>,                       // foundation multisig
    slash_id: u64,
    multisig_signatures: BoundedVec<Signature, ConstU32<7>>,
    decision: MultisigDecision,
) -> DispatchResult;
```

Panel selection: on-chain sortition from top-50 stake, excluding (slashed operator, slashing validator, anyone in same operator's coldkey group). Each panelist posts 1% stake bond.

## Caps and rate limits

- **Single-incident cap:** 10% (1000 bps), enforced at extrinsic level.
- **Monthly cumulative cap:** 50% per operator (5000 bps over 30 rolling days), enforced by checking `MonthlySlashAccumulator` storage.
- **Daily cap:** 30% (3000 bps over 24 hours), prevents runaway cascades.
- **Circuit breaker:** if network-wide slashing exceeds 3× 24-hour rolling baseline, `pallet-slashing` enters paused state requiring 5-of-7 multisig + 2-day public delay to resume (IR playbook §8.2 #2).

## Watcher bond and false-claim penalty

```
fn register_watcher(origin: OriginFor<T>, bond_amount: BalanceOf<T>) -> DispatchResult;
                                            // ≥ 1 ETH-equivalent in OROG
```

If watcher's evidence is rejected by panel or auto-validator:

- 1st offense: bond burned (full bond).
- 2nd offense in 90 days: bond × 10 penalty (stake-bonded if registered as both watcher AND operator) + permanent watcher ban.
- Malicious / forged evidence: criminal referral per plan §9.

## Transparency

Every slash + dispute + ratification is on-chain. `governance-tools` provides a public dispute panel UI showing:

- Slash event + fault code + evidence URL.
- Operator dispute filing.
- Panel composition + votes.
- Multisig ratification.
- Final disposition.

## Versioning

`version` carried in `FaultCode` enum reserved variants. Additive only; no breaking changes without runtime upgrade.

## Open questions

- Should sortition exclude operators registered in last N blocks (anti-flash-panel-stack)? Default: yes, exclude operators registered in last 14 days from panel selection.
- Should panel bond be returned with interest if vote aligns with final disposition? Default: yes, small reward (1% of slash amount split among aligned panelists).

# RFC-0006 — Commit-Reveal Sampling Randomness

**Status:** Draft — ratification target end Q3 2026 **Owner:** Verification Lead **Consumers:** `validator-replay`, `validator-watcher`, `pallet-yuma-consensus`, `pallet-slashing`, `chain-indexer` **Cross-cuts:** RFC-0001 (receipt → leaf for selection), RFC-0004 (batch → input to seed), RFC-0005 (slashing for non-reveal)

## Goal

Define the unpredictable, unmanipulable per-epoch randomness that drives validator replay sampling. The seed must be:

- **Bias-resistant** — no single validator can steer it.
- **Predictable in cadence** — gateways and operators know when a sample is finalized.
- **Cheap to verify** — every full node recomputes it in O(N_validators) hash ops.
- **Late-binding** — receipts cannot be selectively retained/discarded by the operator after the seed is known.

## Design rules (from red-team)

1. **Sample rate ≥10% floor** (rule 2). Per-tier ceilings: `edge` 25%, `prosumer` 20%, `cloud-rented` 15%, `dc-standard` 12%, `dc-premium` 10%, `compliance` 10%.
2. **One-epoch commit-reveal delay** (rule 2). Validator commits at epoch E, reveals at E+1; sampling resolves over receipts in epoch E (closed by then).
3. **Stake-weighted Fisher-Yates** over the operator set; not uniform — high-stake operators bear more verifier load proportionally.
4. **Defection penalties scale.** Single miss → 1 epoch emissions; second within rolling 30 epochs → permit revoked via `pallet-yuma-consensus::revoke_validator`.

## On-chain types

```
 #[derive(Encode, Decode, Clone, PartialEq, Eq, RuntimeDebug, TypeInfo)]
pub struct EpochCommitment {
    pub version: u8,                  // 1
    pub epoch_number: u64,            // the epoch the commitment is FOR (i.e., E)
    pub validator_id: AccountId,      // hotkey
    pub commitment_hash: H256,        // BLAKE2-256(preimage || validator_id ||
epoch_number)
    pub submitted_at: BlockNumber,
    pub signature: Signature,         // ed25519 over canonical-encode
}

#[derive(Encode, Decode, Clone, PartialEq, Eq, RuntimeDebug, TypeInfo)]
pub struct EpochReveal {
    pub version: u8,                  // 1
    pub epoch_number: u64,            // same E the commitment was for
    pub validator_id: AccountId,
    pub preimage: [u8; 32],           // 256-bit random
    pub submitted_at: BlockNumber,    // must be in epoch E+1
    pub signature: Signature,
}

#[derive(Encode, Decode, Clone, PartialEq, Eq, RuntimeDebug, TypeInfo)]
pub struct SampleAssignment {
    pub version: u8,                  // 1
    pub epoch_number: u64,            // the epoch whose receipts are being sampled (=
E)
    pub validator_id: AccountId,      // who replays
    pub operator_id: AccountId,       // whose work is replayed
    pub batch_id: H256,               // RFC-0004 batch
    pub receipt_indices: BoundedVec<u32, ConstU32<1024>>,  // indices into batch merkle
leaves
    pub deadline_block: BlockNumber,  // must submit slashing or attest-clean by then
}
```

## Extrinsics

```
 /// Validator commits to a preimage for epoch E. Must arrive before the first block of
E+1.
fn commit_epoch_seed(
    origin: OriginFor<T>,                // validator hotkey
    epoch_number: u64,
    commitment_hash: H256,
) -> DispatchResult;

/// Validator reveals preimage for epoch E. Must arrive in epoch E+1.
fn reveal_epoch_seed(
    origin: OriginFor<T>,
    epoch_number: u64,
    preimage: [u8; 32],
) -> DispatchResult;

/// Any full node can submit a sample assignment derived from the now-known epoch random
/// for any batch in epoch E. Idempotent; first valid submission persisted.
fn finalize_sample(
    origin: OriginFor<T>,
    epoch_number: u64,
    batch_id: H256,
    assignments: BoundedVec<SampleAssignment, ConstU32<256>>,
) -> DispatchResult;
```

`commit_epoch_seed` rejected if (validator not in top-K=128) or (already committed for this epoch) or (epoch already finalized). `reveal_epoch_seed` rejected if (no commitment) or (preimage hash ≠ commitment) or (out of window). `finalize_sample` rejected if assignments do not match the deterministic algorithm below.

## Seed derivation

After the reveal window closes at end of epoch E+1:

```
epoch_random[E] = BLAKE2-256( concat_sorted_by_validator_id(reveals) )
```

Validators who failed to reveal are excluded from the concatenation; their commitments are recorded for slashing accounting but contribute nothing to the random.

If `epoch_random` would have <3 contributing reveals → fall back to `epoch_random[E] = BLAKE2-256(epoch_random[E-1] || E)` AND emit a `RandomnessDegraded` event. Two consecutive degraded epochs trigger `pallet-system::set_safe_mode(true)` per IR runbook 02.

## Per-receipt sample selector

For each `(epoch_number, batch_id, operator_id)`:

```
seed = BLAKE2-256(
    epoch_random[E]      ||
    receipt_merkle_root  ||      // from SettlementBatch (RFC-0004)
    batch_id             ||
    operator_id
)
```

Stake-weighted Fisher-Yates draws from the operator's stake-weighted receipts in the batch:

```
weights[i] = operator_stake[operator_id_of(receipt_i)] / total_operator_stake_in_batch
target_count = ceil(operator_receipts_in_batch * tier_sample_rate)
```

`tier_sample_rate ∈ {0.25, 0.20, 0.15, 0.12, 0.10, 0.10}` keyed off `OperatorTier` (RFC-0002).

Validator-to-sample assignment uses a second draw with stake weighting over the active validator set; each validator gets a quota of receipts to replay roughly proportional to its validator stake, with a floor of 1 receipt per active validator per epoch.

The selection algorithm is normative — any two full nodes must produce the same `SampleAssignment` set given the same chain state. Reference implementation lives in `validator-replay::sampling::fisher_yates_stake_weighted`.

## Defection penalties

| Event | Penalty |
|---|---|
| Failed to commit (epoch E) | 1 epoch of validator emissions forfeit; recorded in `MissedCommits[E][validator]` |
| Failed to reveal (epoch E+1) after committing | 1 epoch of emissions forfeit + reputation decay (−0.05 in Yuma score) |
| 2nd defection within 30 rolling epochs | Validator permit revoked via `pallet-yuma-consensus::revoke_validator`; stake unbonds with the standard 14-day delay |
| Grinding attempt (multiple commits per epoch detected at gossip layer) | 10% slash via RFC-0005 `ValidatorCollusion` fault code |

Defection events emit `ValidatorDefected(epoch, validator_id, reason)` consumed by `validator-watcher` for scoring.

## Storage

```
 StorageMap: EpochCommitments  (epoch, validator) -> EpochCommitment
StorageMap: EpochReveals      (epoch, validator) -> EpochReveal
StorageMap: EpochRandom       epoch              -> H256        // computed at end of
E+1
StorageMap: MissedCommits     epoch              -> BoundedVec<AccountId, ConstU32<128>>
StorageMap: MissedReveals     epoch              -> BoundedVec<AccountId, ConstU32<128>>
StorageDoubleMap: SampleAssignments (epoch, batch_id) -> BoundedVec<SampleAssignment,
ConstU32<256>>
StorageMap: DegradedEpochs    epoch              -> bool
```

Commitments + reveals pruned after T+epoch_length × 4 to bound state growth. `EpochRandom` retained for 90 days (audit + dispute reconstruction).

## Adversary scenarios

1. **Last-revealer grinding.** Last validator to reveal could choose between revealing or aborting to bias outcome. Mitigation: penalty for non-reveal scales (loss of 1 epoch emissions ≈ ≥10× expected grinding benefit at any realistic stake). Top-K=128 means a single defector shifts ~1/128 of the entropy mass.
2. **Coalition non-reveal.** A coalition of M validators refuses to reveal hoping to force the degraded fallback (which depends on previous epoch). Mitigation: degraded fallback exists but second consecutive degraded epoch trips safe-mode; coalition cost is total emission forfeit.
3. **Commitment grinding via fake identities.** Solved by stake gating on validator set (top-K) and stake-weighted entropy contribution.
4. **Operator pre-mining receipts to avoid sample.** Receipts in batch (RFC-0004) are committed before E ends and merkle-rooted before E+1's reveals are known. Operator cannot retroactively remove a receipt.
5. **Validator-operator collusion.** Validator skews receipt selection toward operator's "safe" receipts. Mitigation: algorithm is deterministic; deviation is provable on-chain and slashable.

## Performance budget

- Commit extrinsic: ~50 bytes; ~50 µs CPU.
- Reveal extrinsic: ~50 bytes; ~50 µs CPU; verifies preimage against stored commitment.
- Finalize-sample: O(N_receipts_in_batch) Fisher-Yates, ~10K receipts ≈ 5 ms. Bounded by `BoundedVec<_, 256>` assignment cap per call.
- Worst case 128 validators × 2 extrinsics × 360 blocks-per-epoch = 256 randomness extrinsics every 72 minutes, well within block weight.

## Versioning

`version: u8 = 1` at TGE. Increment on any field change. Old commitments still resolvable by old algorithm during one runtime cycle of overlap.

## Open questions

- Whether to use VRFs instead of commit-reveal once a stable VRF beacon (drand mainnet bridge or Sassafras) is ratified. Default: commit-reveal at TGE; migrate to VRF in Phase 4.
- Whether `tier_sample_rate` ceilings should be governance-mutable. Default: yes, behind 5-of-7 multisig + 14-d timelock.

# RFC-0007 — Customer Nonce Anti-Replay Protocol

**Status:** Draft — ratification target end Q3 2026 **Owner:** SDK Lead + Pallet Lead **Consumers:** `customer-sdk-py`, `customer-sdk-ts`, `gateway-router`, `gateway-burn-engine`, `worker-control-plane`, `pallet-nonce-vault`, `pallet-bme` **Cross-cuts:** RFC-0001 (receipt carries `customer_nonce`), RFC-0004 (batch settlement enforces nonce burn)

## Goal

Define the customer-generated nonce that ensures a signed inference receipt cannot be re-spent for CUC credit twice, and the on-chain storage that enforces this across the network. Targets red-team rule 11 (replay protection mandatory at customer→gateway→operator boundary).

## Why customer-generated

The customer is the only party with no incentive to allow replay. Operator-generated nonces invite a colluding operator to mint twice the receipts for the same compute. Gateway-generated nonces invite a colluding gateway. The customer signs the nonce, the operator binds it into the receipt (RFC-0001), and the chain burns it at settlement.

## Wire format

```
 #[derive(Serialize, Deserialize, Clone, PartialEq, Eq)]
pub struct CustomerNonce {
    pub version: u8,                // 1
    pub nonce: H256,                // random 256-bit, generated by SDK
    pub ts_ms: u64,                 // milliseconds since UNIX epoch; SDK-clock
    pub customer_id: AccountId,     // customer's account (sr25519 or ecdsa)
    pub validity_window_ms: u32,    // default 60_000; max 86_400_000 (24h)
    pub signature: Signature,       // ed25519/sr25519 over canonical-encode of above
}
```

Canonicalization: SCALE-encode of `(version, nonce, ts_ms, customer_id, validity_window_ms)` then BLAKE2-256 then sign.

## Client-side generation (SDK)

Both `customer-sdk-py` and `customer-sdk-ts` ship a helper:

```python
# Python
from mining_sdk import Customer
c = Customer(keypair=kp)
nonce = c.fresh_nonce()  # → CustomerNonce with ts_ms = now(), 60s window, signed
resp = c.complete(model="llama-3.1-70b", messages=[...], nonce=nonce)
```

```typescript
// TypeScript
import { Customer } from "@mining/sdk";
const c = new Customer({ keypair });
const nonce = c.freshNonce();
const resp = await c.complete({ model: "llama-3.1-70b", messages: [...], nonce });
```

If the caller omits `nonce`, the SDK auto-generates one transparently. Nonces are cryptographic random (`csprng_fill` / `crypto.getRandomValues`). The SDK refuses to reuse a `(nonce, customer_id)` pair within the same process; in-memory LRU prevents accidental retries from re-spending.

## Gateway-side validation

On receiving an inference request from a customer:

1. Verify `signature` against `customer_id` registered key (or accept ephemeral keys for anonymous flows — see §Anonymous customers).
2. Verify `now() - ts_ms < validity_window_ms`; reject if expired (clock skew tolerance ±120s).
3. Reject if `validity_window_ms > 86_400_000` (24h hard cap).
4. Query gateway's local Bloom filter + sliding hash set for `(customer_id, nonce)`; reject if seen.
5. Forward to operator with nonce attached.
6. On receiving signed receipt back: confirm operator embedded `customer_nonce == nonce`.
7. Insert `(customer_id, nonce, ts_ms)` into gateway's nonce vault.

## Operator-side validation

The operator runs an identical Bloom-filter + sliding hash set keyed `(operator_id, customer_nonce)`:

```
struct OperatorNonceVault {
    bloom: BloomFilter<H256>,         // ~10M entries, 0.1% FP
    recent: HashMap<H256, u64>,       // exact set within last 24h, with ts_ms
}
```

Operator MUST reject any inference whose `customer_nonce` is already in `recent` for that customer. The 24h sliding window matches the on-chain enforcement window (next section). Memory budget ~256 MiB per operator.

If the operator's local check passes but the on-chain check at settlement fails (double-submit raced between two gateways), the second receipt's mint is reversed via `pallet-bme::reverse_mint` and the operator is **not** slashed (cooperative behavior; chain wins the tiebreak).

## On-chain storage: `pallet-nonce-vault`

```
 #[derive(Encode, Decode, Clone, PartialEq, Eq, RuntimeDebug, TypeInfo)]
pub struct ShortNonceRecord {
    pub short_hash: [u8; 16],         // BLAKE2-128(customer_id || nonce)
    pub burned_at_block: BlockNumber,
}

StorageMap: RecentNonces  short_hash -> burned_at_block

// Pruning: every block, drop entries where now - burned_at_block > NoncePruneWindow
const NoncePruneWindow: BlockNumber = 14_400;  // ~24h at 6s/block
```

Storing 16-byte short hashes (not full 32-byte nonces) is sufficient for collision resistance over a 24h window: at peak load ~100M requests/day, P(collision) ≈ 1e-13 per pair, negligible.

## Settlement-time burn

When `pallet-job-market::submit_batch` (RFC-0004) processes a `SettlementBatch`:

```
 for receipt in batch_receipts_decoded {
    let short = blake2_128(receipt.operator_id, receipt.customer_nonce);
    ensure!(!RecentNonces::contains_key(&short), Error::NonceAlreadyBurned);
    RecentNonces::insert(&short, current_block);
}
```

Batch rejected wholesale if any nonce is already burned. Gateway must re-submit with the offending receipt removed; gateway slashed under RFC-0005 `FakeBurn` if the gateway knowingly submitted a duplicate (provable by comparing the two batches sharing the nonce).

## Replay attack model

**Attacker:** A malicious actor who has captured a customer's signed nonce + the operator's resulting receipt blob (e.g., from a leaked operator log, a man-in-the-middle on an unencrypted backhaul, or a colluding gateway).

**Attack:** Replay the same `(nonce, receipt)` pair to a second gateway, attempting to: - (a) make the second gateway burn CUC credit a second time, **or** - (b) cause the operator's mint to be credited again.

**Defense path:** 1. Second gateway's local vault may not have seen this nonce → check passes locally. 2. Second gateway forwards to operator. Operator's local vault HAS seen it → operator rejects. 3. If somehow it propagates to settlement (different operator collusion), chain check at `submit_batch` rejects because `short_hash` exists in `RecentNonces`. Whole batch rejected. 4. If attacker tries within 24h+1 second to evade the window → fails because `validity_window_ms ≤ 24h` and `ts_ms` is signed by customer.

CUC double-spend is prevented at three layers (gateway, operator, chain). Operator double-mint is prevented at chain. Defense in depth.

## Anonymous customers

For the burn-engine SaaS gateway (MiningLabs Inc. pre-TGE), customers may use ephemeral nonce keys. The gateway then signs the nonce as a delegated signer:

```
pub struct DelegatedCustomerNonce {
    pub inner: CustomerNonce,
    pub gateway_id: AccountId,         // who is taking on accountability
    pub gateway_signature: Signature,
}
```

The gateway absorbs replay risk and the underlying `customer_id` is the gateway's pseudonymous customer-pool key. Burn vs. mint accounting unchanged.

## Bandwidth

- On-chain storage growth: peak 100M nonces/day × 24h × 16 bytes = 38 GiB rolling window. Pruned daily; archive nodes retain.
- Per-block prune work: O(blocks_per_day_window_slice); ~7K ops/block worst case, ~3 ms.
- Bloom filter operator-side: ~256 MiB RAM, lookup O(1).

## Extrinsics

```
/// Read-only helper (not strictly necessary; included for SDK convenience).
fn is_nonce_burned(short_hash: [u8; 16]) -> bool;

/// Internal — called inside submit_batch's pipeline; not user-callable.
fn burn_nonces(batch_id: H256, short_hashes: Vec<[u8; 16]>) -> DispatchResult;
```

There is no standalone user extrinsic for nonce burn — burns are atomic with `submit_batch` (RFC-0004) so they cannot be separated from the mint they authorize.

## Errors

| Error | When |
|---|---|
| `NonceAlreadyBurned` | Chain saw this short_hash in window |
| `NonceExpired` | `now() - ts_ms > validity_window_ms` |
| `NonceWindowTooLong` | `validity_window_ms > 86_400_000` |
| `NonceSignatureInvalid` | Customer signature does not verify |
| `NonceCustomerNotRegistered` | (Permissioned-mode only) customer not in allowlist |

## Versioning

`version: u8 = 1` at TGE. Increment on field changes; previous-version nonces accepted for one runtime cycle.

## Open questions

- Whether to publish a customer-side "spent" subscription (chain → SDK) so wallets can detect their nonces being used. Default: yes — chain emits `NonceBurned(short_hash, burned_at_block)` events the SDK can filter for `short_hash` it generated.
- Whether the 24h window should be governance-mutable. Default: yes, behind 5-of-7 multisig + 7-d timelock.

# RFC-0008 — Price Oracle Feed for BME Settlement

**Status:** Draft — ratification target end Q3 2026 **Owner:** Tokenomics Lead + Pallet Lead **Consumers:** `pallet-oracle-twap`, `pallet-bme`, `gateway-burn-engine`, `validator-watcher` **Cross-cuts:** RFC-0004 (batch settlement reads CurrentTwap), RFC-0005 (oracle defection is slashable)

## Goal

Provide a manipulation-resistant USD price feed for OROG and CUC, used by `pallet-bme` to compute the burn:mint ratio at settlement. Targets red-team rule 10: long-window TWAP at launch to prevent flash-loan / first-listing manipulation.

## Sources (4)

1. **Binance** — top-CEX price for the OROG/USDT pair (post-listing).
2. **Coinbase** — second top-CEX for OROG/USDC.
3. **On-chain DEX (Uniswap V3 on Ethereum)** — OROG/USDC pool, observed via Snowbridge → `pallet-oracle-twap` adapter (post-listing).
4. **Uniswap V4 protocol-managed AMM** — bonded foundation-controlled hook-AMM on our own chain (launched at TGE, holds protocol liquidity).

Source set is governance-mutable behind 5-of-7 multisig + 14-d timelock + 30-d cool-down.

## Aggregation algorithm

Per epoch (or finer):

```
 1. Collect all source prices submitted in the lookback window.
2. Compute TWAP per source: weighted by time-segment length.
3. Compute the median across the 4 source TWAPs.
4. Compute deviation of each source TWAP from the median.
5. Exclude any source where |dev| > 5% of median.
6. If remaining sources < 2 → fallback (see below).
7. Aggregate TWAP = stake-weighted mean of remaining sources.
       (oracle pool stake-weight; foundation pool default-equal)
8. Write CurrentTwap.
```

## Lookback window

Per red-team rule 10:

| Phase | Window | Notes |
|---|---|---|
| Months 0–18 post-TGE | 4–12 hours | Tunable by oracle ops within bounds; default 8h. Long window blunts flash-loan and thin-market attacks at fresh listings. |
| Months 19+ | 30 minutes | Step-down via governance proposal; cannot be set <30 min without a runtime-upgrade RFC. |

Window changes require 5-of-7 multisig + 14-d timelock; no emergency reduction below 30 min.

## On-chain types

```
 #[derive(Encode, Decode, Clone, PartialEq, Eq, RuntimeDebug, TypeInfo)]
pub struct PriceSample {
    pub version: u8,                  // 1
    pub source_id: u8,                // 0=Binance 1=Coinbase 2=UniV3-ETH 3=UniV4-local
    pub price_usd_q64: u128,          // OROG→USD as fixed-point Q64.64
    pub volume_usd_q64: u128,         // 0 for AMM TWAP; nonzero for CEX samples
    pub ts_ms: u64,
    pub oracle_id: AccountId,         // submitter
    pub signature: Signature,
}

#[derive(Encode, Decode, Clone, PartialEq, Eq, RuntimeDebug, TypeInfo)]
pub struct AggregatedTwap {
    pub version: u8,
    pub computed_at_block: BlockNumber,
    pub window_start_ms: u64,
    pub window_end_ms: u64,
    pub price_usd_q64: u128,
    pub contributing_sources: BitFlags8,
    pub excluded_sources: BitFlags8,    // those clipped as outliers
    pub fallback: bool,                 // true if static-fallback active
}

#[derive(Encode, Decode, Clone, PartialEq, Eq, RuntimeDebug, TypeInfo)]
pub struct OraclePoolMember {
    pub oracle_id: AccountId,
    pub stake: BalanceOf<T>,           // separate stake pool; min 10× operator min
stake
    pub registered_at: BlockNumber,
    pub good_submissions: u64,
    pub bad_submissions: u64,
    pub last_submitted_at: BlockNumber,
}
```

## Storage

```
 StorageMap:   OraclePool       AccountId -> OraclePoolMember
StorageMap:   PriceHistory     (source_id, ts_bucket) -> BoundedVec<PriceSample,
ConstU32<32>>
StorageValue: CurrentTwap      AggregatedTwap
StorageValue: WindowMs         u64              // current TWAP window
StorageValue: EmergencyPaused  bool             // multisig kill-switch
StorageValue: LastGoodPrice    AggregatedTwap   // for static fallback
```

`PriceHistory` is pruned beyond `WindowMs × 2` lookback.

## Extrinsics

```
 /// Oracle pool member submits a per-source price sample.
fn submit_price(
    origin: OriginFor<T>,            // must be in OraclePool
    sample: PriceSample,
) -> DispatchResult;

/// Recompute the aggregated TWAP. Permissionless callable but rate-limited;
/// also auto-called on every batch settlement via on_initialize.
fn refresh_twap(origin: OriginFor<T>) -> DispatchResult;

/// 5-of-7 multisig: emergency pause minting. Freezes pallet-bme mint path
/// but does NOT halt burns or operator emissions from prior epochs.
fn emergency_pause(origin: OriginFor<T>) -> DispatchResult;

fn emergency_unpause(
    origin: OriginFor<T>,
    multisig_signatures: BoundedVec<Signature, ConstU32<7>>,
) -> DispatchResult;

/// Governance: rotate oracle pool membership.
fn add_oracle(origin: OriginFor<T>, oracle_id: AccountId, stake: BalanceOf<T>) ->
DispatchResult;
fn remove_oracle(origin: OriginFor<T>, oracle_id: AccountId, reason_hash: H256) ->
DispatchResult;
```

## Oracle pool

- ≥5 active oracle members at all times. Below 5 → `emergency_pause` auto-triggered by `on_initialize`.
- Stake: ≥10× operator minimum (denomination follows DECISIONS.md H9 default — OROG with USD-pegged ratchet every 30 days).
- Rotation: term-limited 90 days; minimum 30 days between re-add by same operator.
- Geo-distributed: ≥3 distinct jurisdictions among the active set.
- Disjoint from validators and gateways at TGE.

## Fallback logic

| Trigger | Action |
|---|---|
| ≥2 outlier sources clipped in one TWAP cycle | Use `LastGoodPrice` (≤ 24h old); emit `OracleFallbackActive` |
| Source unreachable for >30 min | Mark source `inactive`; auto-redrop after 1h healthy heartbeat |
| 3 of 4 sources unreachable | `emergency_pause()` auto-triggered; multisig must `emergency_unpause` |
| `LastGoodPrice` older than 24h AND no live aggregate | Halt mint path entirely; burns still allowed; IR runbook 03 |

`pallet-bme::emergency_pause()` is the user-visible kill-switch; gated 5-of-7 multisig.

## Defection penalties

| Behavior | Penalty |
|---|---|
| Submitted price ≥10% from final agg (per-submission outlier) | Increment `bad_submissions`; reputation decay |
| `bad_submissions / total ≥ 20%` over rolling 30 days | Auto-eject from pool; 25% stake slash (`FakeBurn` analog routed through RFC-0005 with custom code `OracleManipulation`) |
| Failed to submit for >12 hours while in pool | Soft penalty: half-emission for that epoch |
| Provable collusion (price agreement off-chain leaked) | 100% slash; permanent ban |

Slashing for oracle pool flows through RFC-0005 with the new fault-code variant `OracleManipulation` (severity 2500 bps base, scalable).

## Manipulation cost analysis

To shift the aggregated TWAP by >5% with the 8h window:

- Attacker must hold majority position in ≥3 of 4 sources for ≥4h continuous.
- For UniV4 protocol-managed AMM: liquidity depth is foundation-controlled at TGE (≥$5M) — attacker needs to absorb that depth.
- For CEX sources: 4h volume-weighted shift requires multi-million-dollar sustained pressure; CEX surveillance flags it.
- Cost asymmetry: oracle stake slash + multi-source manipulation cost >> potential mint inflation gain because BME caps headroom independently.

## Bandwidth

- 4 sources × 1 sample/source/min × 4 oracle pool members = 16 extrinsics/min nominally.
- ~96 bytes/sample × 16 × 60 × 24 = 2.1 MiB/day on-chain.
- Pruning maintains `PriceHistory` at ~1 day worst case.

## Versioning

`version: u8 = 1` at TGE. Source matrix governance-mutable; window governance-mutable bounded. Breaking changes require RFC bump.

## Open questions

- Whether to add Chainlink as a 5th source. Default: no at TGE; consider Q3 2027 once their cross-chain pricing rails are mature on Snowbridge.
- Whether to publish source-level TWAPs publicly. Default: yes — `status-page` shows source TWAPs + agg + clipped flags real-time.

# RFC-0009 — Operator Registration Flow

**Status:** Draft — ratification target end Q3 2026 **Owner:** Pallet Lead + Compliance Lead **Consumers:** `pallet-operator-stake`, `pallet-attestation-registry`, `sanctions-screener`, `operator-onboarding-ui`, `gateway-router` **Cross-cuts:** RFC-0002 (attestation matrix per tier), RFC-0003 (heartbeat capabilities), RFC-0005 (collision = slash), RFC-0008 (TWAP for USD-pegged stake)

## Goal

Define how an operator joins the network: which artifacts they must submit, how the chain validates them, what determines their tier, and how upgrades/downgrades flow. Targets red-team rules 7 (device-cert collisions), 16 (sanctions screening at onboarding).

## Identity hierarchy

```
coldkey   — long-term holding; stakes; cannot serve traffic directly
└── hotkey — operational signer for receipts, heartbeats, slashing extrinsics
    └── device_cert — silicon-bound (NVIDIA Device Identity CA), 1:1 with GPU
```

One coldkey may register multiple hotkeys (multi-operator). One hotkey is 1:1 with one device_cert at any time. Re-binding requires deregistration first.

## Tiers (recap from RFC-0002)

| Tier | Attestation requirement | Min stake (USD-pegged) | Notes |
|---|---|---|---|
| `dc-premium` | NVIDIA CC + (TDX OR SEV-SNP) | $5,000 | Highest emission share |
| `dc-standard` | NVIDIA CC + (TDX OR SEV-SNP) | $2,500 | Standard data-center |
| `cloud-rented` | NVIDIA CC + Intel TDX | $2,500 | Cloud GPU; H100/H200 |
| `prosumer` | none (stake-only) | $2,000 | Best-effort tier |
| `edge` | none (stake-only) | $1,500 | Edge / consumer GPUs |
| `embed-only` | none | $1,000 | Whisper / SD / small models |
| `compliance` | NVIDIA CC + TDX + SEV-SNP + SOC 2 | $25,000 | HIPAA/PCI workloads |

Stake denominated per DECISIONS.md H9 default: **OROG with USD-pegged ratchet via 5-of-7 multisig governance every 30 days, using `CurrentTwap` (RFC-0008) as reference price**.

## Registration extrinsic

```
 #[derive(Encode, Decode, Clone, PartialEq, Eq, RuntimeDebug, TypeInfo)]
pub struct OperatorRegistration {
    pub version: u8,                              // 1
    pub coldkey: AccountId,
    pub hotkey: AccountId,
    pub attestation_quote: AttestationReportHash, // == OnChainAttestation.report_hash,
RFC-0002
    pub tier: OperatorTier,
    pub stake_amount: BalanceOf<T>,               // in OROG native units
    pub geo_region: BoundedString<8>,             // ISO-3166-1 alpha-2 + subdivision
    pub ip_24_hash: H256,                         // BLAKE2(public IP /24) for diversity
enforcement
    pub sanctions_check_proof: SanctionsCheckProof,
    pub coldkey_signature: Signature,             // over canonical-encode(everything
above)
    pub hotkey_signature: Signature,              // separate, proves hotkey control
}

#[derive(Encode, Decode, Clone, PartialEq, Eq, RuntimeDebug, TypeInfo)]
pub struct SanctionsCheckProof {
    pub version: u8,
    pub provider_id: u8,                          // 0=Chainalysis 1=TRM 2=Elliptic
    pub screened_address: AccountId,              // must == coldkey
    pub screened_at_ts: u64,                      // must be ≤ 24h before tx
    pub passed: bool,                             // must be true to register
    pub screener_signature: Signature,            // signed by `sanctions-screener`
service key
}

fn register_operator(
    origin: OriginFor<T>,                         // any signer; bound to coldkey via
signature
    registration: OperatorRegistration,
) -> DispatchResult;
```

## Validation steps

In `pallet-operator-stake::register_operator`:

1. **Signatures.** Verify `coldkey_signature` against `coldkey`; verify `hotkey_signature` against `hotkey` (separate-key proof-of-control).
2. **Attestation.** Look up `attestation_quote` in `pallet-attestation-registry`. Must be: - Not expired ( `expires_at > now` ). - Not revoked. - `vendor_set` satisfies the tier requirement (RFC-0002 matrix). - `OnChainAttestation.operator_id == hotkey`.
3. **Device-cert collision (rule 7).** Look up `OnChainAttestation.gpu_uuid` across all existing registrations. - If found AND under a different coldkey → **both coldkeys slashed 100%** via `pallet-slashing::submit_slashing_evidence(..., FaultCode::DeviceCertCollision, ...)`. No grace, no dispute. - If found under same coldkey → reject as duplicate registration.

4. **Stake.** `stake_amount ≥ tier_min_stake_useful()` where the per-tier minimum is computed at extrinsic time from `CurrentTwap`. Stake transferred from `coldkey` to `pallet-operator-stake` reserved balance.
5. **Sanctions.** Verify `sanctions_check_proof`: - `screener_signature` signed by an authorized screener key (allowlist in `pallet-treasury-ext`). - `screened_at_ts` within 24h. - `passed == true`. - `screened_address == coldkey`. - Reject otherwise. Cached "passed" results are not accepted from a third party.
6. **Geo / IP diversity.** Check `(geo_region, ip_24_hash)` against `OperatorDiversityCaps`: - No more than 5% of total active operators in any single `ip_24_hash`. - No more than 20% in any single `geo_region`. - Soft caps: registration accepted but flagged `diversity_warning=true`; emission share reduced 20% while over cap.
7. **Tier-stake binding.** Insert into `OperatorRecords` map; emit `OperatorRegistered(coldkey, hotkey, tier, attestation_quote)`.

## Storage

```
 #[derive(Encode, Decode, Clone, PartialEq, Eq, RuntimeDebug, TypeInfo)]
pub struct OperatorRecord {
    pub coldkey: AccountId,
    pub hotkey: AccountId,
    pub tier: OperatorTier,
    pub stake_useful: BalanceOf<T>,
    pub attestation_report_hash: H256,
    pub geo_region: BoundedString<8>,
    pub ip_24_hash: H256,
    pub registered_at: BlockNumber,
    pub last_heartbeat_at: BlockNumber,
    pub status: OperatorStatus,         // Pending | Active | Suspended | Deregistering
| Slashed
    pub diversity_warning: bool,
}

StorageMap:        OperatorRecords     hotkey   -> OperatorRecord
StorageMap:        ColdkeyOperators    coldkey  -> BoundedVec<AccountId,
ConstU32<32>>  // hotkeys
StorageMap:        GpuUuidIndex        H256     -> AccountId  // hotkey owning this GPU
StorageMap:        Ip24Counts          H256     -> u32
StorageMap:        GeoRegionCounts     BoundedString<8> -> u32
StorageValue:      DiversityCaps       DiversityCapsConfig
```

## Tier upgrade / downgrade

```
fn upgrade_tier(
    origin: OriginFor<T>,                       // coldkey
    hotkey: AccountId,
    new_tier: OperatorTier,
    additional_stake: BalanceOf<T>,             // may be 0 if existing stake covers new
min
    new_attestation_quote: Option<H256>,        // required if new tier has stricter
attestation
) -> DispatchResult;

fn downgrade_tier(
    origin: OriginFor<T>,
    hotkey: AccountId,
    new_tier: OperatorTier,
) -> DispatchResult;
```

Upgrade is effective immediately if attestation + stake satisfy new tier requirements. Downgrade enters a 7-day cool-down where the operator continues to serve at the lower tier; any pending receipts settle at the previous tier's pricing.

## Deregistration

```
fn deregister_operator(
    origin: OriginFor<T>,                       // coldkey
    hotkey: AccountId,
) -> DispatchResult;
```

Flow: 1. Operator status → `Deregistering`. 2. New jobs no longer routed (gateway-side respects the status flag). 3. 14-day unbonding period; stake held in escrow. 4. After 14d, stake returned to coldkey; record archived; `GpuUuidIndex` entry cleared. 5. Any slashing extrinsic accepted during the 14d window resolves first.

## Sanctions re-screening

`sanctions-screener` re-screens every active operator's coldkey: - Every 24h continuously. - Within 1h of any inbound transfer >$10K equivalent. - Real-time on outbound transfers.

A failed re-screen emits `SanctionsHit(coldkey)`; `pallet-slashing` consumes via RFC-0005 `SanctionsHit` fault code (100%, frozen — not burned).

## Hotkey rotation

```
fn rotate_hotkey(
    origin: OriginFor<T>,                       // coldkey
    old_hotkey: AccountId,
    new_hotkey: AccountId,
    new_attestation_quote: H256,                // new key bound to same GPU
    coldkey_signature: Signature,
    new_hotkey_signature: Signature,
) -> DispatchResult;
```

Rotation must point to the same `gpu_uuid` to prevent a coldkey from accumulating extra silicon stealthily. `GpuUuidIndex` updated atomically.

## Errors

| Error | When |
|---|---|
| `AttestationInvalid` | quote not in registry, expired, revoked, or wrong vendor_set |
| `DeviceCertCollisionDetected` | gpu_uuid already exists under different coldkey (also triggers slash) |
| `StakeBelowTierMinimum` | stake_amount < tier_min_stake_useful() at TWAP |
| `SanctionsCheckFailed` | screener says fail, or proof invalid/stale |
| `DiversityCapExceeded` | hard cap (not soft warning) hit |
| `TierNotEligibleForRegistration` | tier=embed-only requested but model registry says base model unsupported |
| `ColdkeyHotkeyAlreadyBound` | duplicate registration |

## Versioning

`version: u8 = 1` at TGE. Increment on field changes; one runtime cycle of mixed-version tolerance.

## Open questions

- Whether to allow re-registration after slash + unbond + re-screen, or permanent ban. Default: re-registration allowed after 6 months + new coldkey + fresh sanctions screen — except for `SanctionsHit` (permanent) and `DeviceCertCollision` (permanent for affected coldkey).
- Whether geo-region diversity caps should be governance-mutable. Default: yes, behind 5-of-7 multisig + 14-d timelock.

# RFC-0010 — RPC Endpoint Provider Contract

**Status:** Draft — ratification target end Q3 2026 **Owner:** Infra Lead + Foundation Ops **Consumers:** `chain-node`, `gateway-router`, `worker-control-plane`, `wallet-sdk-core`, `customer-sdk-{py,ts}`, `explorer-web`, `status-page` **Cross-cuts:** RFC-0003 (operator heartbeats RPC), RFC-0004 (batch submission RPC), RFC-0008 (oracle submission RPC)

## Goal

Ensure no single party can de-platform the network at the RPC layer. Targets red-team rule 12: at TGE, 3 independent providers run public Substrate JSON-RPC + Frontier EVM JSON-RPC endpoints with published SLA on a foundation-neutral status page.

## Provider topology at launch

| Provider | Operator | Region | Role |
|---|---|---|---|
| `rpc.mining.network` | Foundation-operated | US-East + EU-West dual | Reference (foundation runs 1) |
| `rpc.figment-class.mining.network` | Partner A (Figment-class) | NA + APAC | Independent |
| `rpc.allnodes-class.mining.network` | Partner B (Allnodes-class) | EU + APAC | Independent |

"Independent" means: separate legal entity, separate infrastructure account, separate Layer-1 hosting (not all on AWS). Foundation cannot revoke a partner's keys; partner cannot revoke foundation's.

The hostnames above are templates — exact provider names finalized at partner-onboarding.

## Required surface

Each provider commits to the full surface:

### Substrate JSON-RPC (mandatory)

- `chain_*`, `state_*`, `system_*`, `payment_*`, `author_*`, `rpc_methods`
- Runtime API: `Metadata`, `Core`, all pallet APIs in `runtime-mainnet`
- Custom RPC methods registered by pallets: `oracle_*`, `attestation_*`, `nonce_*`, `slashing_*` (these resolve to read-only chain state inspectors)

### Frontier EVM JSON-RPC (mandatory)

- `eth_*` standard methods (Metamask-compatible subset)
- `net_*`, `web3_*`
- `eth_subscribe` (newHeads, logs, newPendingTransactions)

### WebSocket subscriptions (mandatory)

- All chain subscriptions: `chain_subscribeNewHeads`, `chain_subscribeFinalizedHeads`, `state_subscribeStorage`
- All EVM subscriptions

### Archive node (mandatory)

- Full state at every block since genesis. No pruning. Required for explorer + dispute reconstruction.

### Off-chain CDN (mandatory)

- Serve receipt blobs for any `(batch_id, receipt_hash)` referenced by a batch the provider witnessed within the last 90 days.

## Method support matrix (excerpt)

| Method | Foundation | Partner A | Partner B | Notes |
|---|---|---|---|---|
| `chain_getBlock` | Y | Y | Y | |
| `state_call` | Y | Y | Y | |
| `state_getStorage` | Y | Y | Y | |
| `state_subscribeStorage` | Y | Y | Y | WS |
| `author_submitExtrinsic` | Y | Y | Y | Rate-limited per-IP |
| `eth_call` | Y | Y | Y | |
| `eth_estimateGas` | Y | Y | Y | |
| `eth_sendRawTransaction` | Y | Y | Y | Rate-limited |
| `eth_subscribe` | Y | Y | Y | WS |
| `oracle_getCurrentTwap` | Y | Y | Y | Pallet RPC |
| `attestation_isRevoked` | Y | Y | Y | Pallet RPC |
| `nonce_isBurned` | Y | Y | Y | Pallet RPC |
| `mining_replayReceipt` | Y | Optional | Optional | Heavy; foundation always supports |

## SLA targets

| Metric | Target | Measurement window | Penalty for miss |
|---|---|---|---|
| Uptime | 99.9% | Rolling 30 days | Demerit; 3 demerits in 90d → provider replaced |
| p99 query latency (cached read) | <500 ms | Rolling 24h | Public status flag |
| p99 query latency (state_call) | <2 s | Rolling 24h | Public status flag |
| WebSocket pub/sub delivery | <1 s | Rolling 24h | Public status flag |
| Block-tip freshness | within 3 blocks of chain head | Real-time | Status auto-marks degraded |
| Archive completeness | 100% | Audit monthly | Replacement on miss |
| CDN receipt availability | 99.5% | Rolling 30d | Demerit |
| Incident response | acknowledged <15 min for P0 | Per-incident | Demerit |

The `status-page` publishes uptime % and live latency per provider; greenwashing is impossible because the same data is verifiable by anyone querying the endpoint.

## Rate limits

Per-IP defaults (provider may relax for authenticated keys):

| Endpoint class | Limit |
|---|---|
| Read RPC (HTTP) | 100 req/s, 10K req/min |
| Read RPC (WebSocket subs) | 100 active subs/connection |
| Write RPC ( `author_submitExtrinsic` , `eth_sendRawTransaction` ) | 10 req/s |
| Heavy RPC ( `mining_replayReceipt` ) | 1 req/s per key |
| Archive deep-state queries | 10 req/s |

CORS: - Mainnet endpoints: `Access-Control-Allow-Origin: *` for read-only methods; write endpoints restricted to a documented allowlist of frontend origins; non-origin'd POST requests always allowed (CLI/SDK). - Testnet endpoints: `Access-Control-Allow-Origin: *` everywhere.

## Operator-daemon fallback logic

```
 # pseudocode — implemented in worker-control-plane
def rpc_client():
    primary, secondary, tertiary = load_rpc_pool()
    cur = primary
    while True:
        try:
            ws = connect(cur, timeout=2)
            while True:
                if last_block_age(ws) > 120:        # 2 minutes
                    raise StaleRpc(cur)
                yield ws
        except (StaleRpc, ConnError):
            cur = next_after(cur, [primary, secondary, tertiary])
            if cur is primary:                       # we've cycled through all
                spawn_local_validator_node_cold_sync(target_minutes=60)
                return local_ws
```

Targets: - Primary RPC failover within 2 minutes of unresponsiveness. - Tertiary failover within 4 minutes. - All-three-down → operator runs its own validator node with 1h cold-sync target (uses pinned snapshots from `weight-cdn-pinner`).

## Mainnet vs. testnet endpoints

| Network | Hostnames | Notes |
|---|---|---|
| Mainnet | `rpc.mining.network`, `wss.mining.network`, `evm.mining.network` | Behind multi-region anycast |
| Testnet (Forge) | `rpc.forge.mining.network`, etc. | Faucet integrated |
| Testnet (Forge-Stake) | `rpc.forge-stake.mining.network` | KYC'd, incentivized |
| Testnet (Forge-Adversarial) | `rpc.forge-adv.mining.network` | Permissionless attack net |
| Shadowfork | `rpc.shadow.mining.network` | Pre-TGE final check; not public |

DNS records owned by foundation; SLA contracts include "right to transfer DNS to a successor provider on 30-d notice" — prevents provider lock-in.

## Partner SLA contract template (key clauses)

1. **Term.** 24 months initial, 12-month renewals.
2. **Compensation.** Per-month base fee + per-million-request bonus; bonuses pro-rated by SLA achievement.
3. **Right to audit.** Foundation may inspect provider logs (subject to GDPR redaction) and synthetic monitoring data.

4. **Method coverage attestation.** Partner publishes monthly attestation that all mandatory methods return correct results (signed by partner key).
5. **Key custody.** Provider holds operational keys; no slashing exposure (read/archive providers are not consensus validators).
6. **Termination triggers.** Three demerits in 90 days; material breach (intentional method removal); regulatory action against provider entity.
7. **Wind-down.** 60-day handoff to successor provider; provider continues service through handoff.
8. **Independence covenant.** Provider may not be acquired by foundation or any C-corp affiliate during the term.
9. **Transparency.** Provider's status page is publicly readable; uptime claims are independently measurable.
10. **Indemnification.** Mutual indemnity for negligence; provider not liable for chain-level malfunctions.

## Status page

`status-page` aggregates: - Provider uptime % (live). - p99 query latency per provider (live). - WebSocket lag per provider (live). - Block-tip freshness per provider (live). - Archive completeness audit results (monthly). - Incident log with timestamped impact + resolution. - Demerit log (visible). - Method support matrix snapshot.

`status-page` is foundation-operated but the data is queryable directly from each provider; status-page just aggregates and graphs.

## DDoS / abuse handling

Each provider commits to: - Cloudflare-class WAF in front of HTTP endpoints. - IP-rate-limiting + per-API-key rate-limiting at edge. - Anycast routing for the WebSocket fleet. - Failover within the provider's own pool independent of the cross-provider failover.

If an attack overwhelms a provider, the daemon failover (above) routes operators to remaining providers. If 2 of 3 providers degrade simultaneously (active attack), `chain-node` daemons fall back to running embedded validator nodes; the network keeps producing blocks because validators run on a separate private peer network not exposed via public RPC.

## Versioning

`version` not encoded on-chain (this is operational, not on-chain state). Method support matrix versioned in `chain-tooling-rust/specs/rpc-matrix.toml` (added in companion PR).

## Open questions

- Whether to require a 4th provider before mainnet (defense in depth). Default: 3 at TGE, target 5 by end of Year 1.

- Whether to publish provider keys on-chain so they can sign their own status attestations verifiably. Default: yes, registered in `pallet-treasury-ext::RpcProviders` map with rotation timelock.