Experimental Staking Module — Implementation Phase (Lean MVP) for kBTC

To: Kintsugi / Interlay Community
Date: August 10, 2025

> **Preface — architectural direction (kept concise)**
> We initially assessed a native Substrate pallet approach. Given the ecosystem's **shift** toward **JAM**, broad **EVM compatibility**, and near-term **PolkaVM** prospects, we prioritize an **EVM-first** module with a clear path to PolkaVM.

---

1) Executive summary

This document specifies a **Lean MVP (Phase 1)** for the **kBTC staking module**, intentionally scoped to be **feasible within ~3 months** and **within a $25k budget**. The module enables **staking kBTC to accrue "Points"** and, optionally, to **mint an external ERC-20 reward token** if an address with `MINTER_ROLE` is provided. Heavy features (slashing, complex oracles, dynamic bonding curves, full governance integrations) are **deferred to Phase 2**.

Lean MVP outcomes:

- **On-chain:**
  - `StakingVault` (kBTC lock/unstake with cooldown).
  - `PointsEmitter` with **fixed-rate emissions** per block or per second, accruing to stakers **pro rata**.
  - Optional `ExternalTokenMinter` adapter to mint rewards on a provided ERC-20 (role-gated).

- **Client:** minimal **TypeScript SDK** (stake/unstake/claim, read accruals).
- **UI (alpha-lite):** single-page stake/unstake/claim + balance/points view.
- **Quality:** unit & integration tests; **coverage ≥ 80%** on core; Slither static analysis; basic invariant checks (no reentrancy, conservation of value).
- **Ops:** testnet deployment scripts/configs; short **runbook** (emergency pause, parameter changes); basic dashboard.

Out of scope (Phase 1): **Slashing**, **RiskOracle**, **dynamic curves**, **governance adapters/timelocks**, **subgraph**, complex monitoring.

---

2) Context and strategic motivation

- **Why EVM-first:** faster iteration, better tooling, immediate **dApp** composability; PolkaVM kept as a forward path via interface abstraction.
- **Why Lean MVP:** deliver a usable **staking-to-points/rewards** primitive quickly, collect telemetry and user feedback, then graduate to Phase 2.

---

3) Purpose and value proposition

- **Utility for kBTC:** non-custodial staking that **generates Points** and, optionally, **external rewards**.
- **Low-friction integrations:** dApps can read Points for boosts, allowlists, fee tiers, or governance weight; or plug an ERC-20 for direct incentives.
- **Replicability:** simple primitives that other parachains can reuse.

---

4) Implementation scope — what will be delivered (Phase 1)

1. **Contracts (Solidity)**
   - `StakingVault`: deposit/withdraw, cooldown, per-user accounting, pause guard.
   - `PointsEmitter`: fixed emission rate; pro rata accrual by stake-time; claim to **non-transferable Points balance** (event + mapping).
   - `ExternalTokenMinter` (optional): if configured with an ERC-20 that grants minter rights, mirror claims as token mints.
   - **Admin:** set emission rate, pause/unpause, set cooldown; owner as **2-of-3 multisig** (placeholder).
2. **SDK (TypeScript)**
   - Functions: `stake()`, `requestUnstake()`, `executeUnstake()`, `claimPoints()`, `claimExternal()`, `getAccrued()`; typings and basic examples.
3. **UI alpha-lite**
   - Single screen: stake/unstake/claim; balances; read-only emission rate; warnings on cooldowns/pauses.
4. **Quality & ops**
   - Tests: unit + integration; **coverage ≥ 80%**; basic invariant suite.
   - Static analysis (Slither); gas snapshots for core paths.
   - Scripts for devnet/testnet; minimal runbook (pause, parameter changes, upgrade placeholder).

De-scoped to Phase 2: slashing policies, oracle adapters, dynamic reward curves, timelocked governance, full subgraph/indexing, advanced dashboards.

---

5) Technical approach & minimal architecture

- **Non-custodial:** users hold kBTC in `StakingVault`; accounting uses shares with precision math.
- **Simple emissions:** fixed `emissionPerSecond` (or per block) distributed via a global cumulative index to avoid per-user loops.
- **Points model:** non-transferable by default (on-chain balance + events). Optional mirroring to an external ERC-20 if provided.
- **Upgrade posture:** proxy avoided in Phase 1; instead, small contracts, a clear migration path, and a pause switch.
- **Observability:** events for `Stake`, `UnstakeRequested`, `UnstakeExecuted`, `PointsAccrued`, `PointsClaimed`, `EmissionRateUpdated`, `Paused`.

---

6) Test strategy (lean)

- **Unit/integration:** deposit/withdraw flows, accrual math, cooldown edge cases.
- **Invariants (basic):** conservation of shares, monotonic global index, no reentrancy, no stuck funds.
- **Static analysis:** Slither baseline; manual review checklist.
- **Coverage target: ≥ 80%** on core modules.
- **Gas checks:** deposit/unstake/claim snapshots.

---

7) UI/UX — alpha-lite scope

- Minimal stake/unstake/claim; balances & accrued Points; copy for cooldowns/pauses; hardware-wallet-friendly.
- No governance panels or advanced risk views in Phase 1.

---

8) Timeline & milestones (≈ 12 weeks)

1. **Weeks 1–2 — Kickoff & scaffolding:** finalize lean spec; repos; CI; Slither in CI; minimal design doc.
2. **Weeks 3–5 — Contracts v1:** `StakingVault` + `PointsEmitter`; unit tests; gas baselines.

3. **Week 6 — Optional adapter:** `ExternalTokenMinter`; integration tests.
4. **Weeks 7–8 — SDK + UI alpha-lite:** core flows wired to testnet.
5. **Week 9 — Hardening:** invariants, static-analysis pass, pause drills; runbook v1.
6. **Week 10 — Public testnet RC:** community testing; bug triage.
7. **Weeks 11–12 — RC2 & readiness:** fixes, docs, handover; mainnet checklist (staged caps).

---

9) Budget (fixed, Phase 1) — $25,000

- **Smart contracts (design/build/tests):** $10,000
- **SDK + UI alpha-lite:** $6,000
- **QA & security (tests, Slither, invariants):** $4,000
- **DevOps & testnet ops:** $2,000
- **PM/coordination & contingency:** $3,000

External audits and bug bounties are **out of scope** for this budget and recommended for Phase 2.

---

10) Success metrics (Phase 1)

- Contracts deployed to public testnet; **≥ 50 successful stake/claim transactions** without incidents.
- **Coverage ≥ 80%**; zero **high-severity** static-analysis findings outstanding.
- Emission math verified against a reference model across **≥ 10 scenarios**.
- SDK and UI alpha-lite used by at least **two integration partners** or community testers.
- Mainnet readiness checklist completed with staged caps defined.

---

11) Risks & mitigations

- **Parameter misconfiguration (emission/cooldown):** guarded by a pause switch; testnet telemetry before mainnet.
- **Integration drift:** stable SDK; examples; semantic versioning.
- **Upgrade needs:** small contracts + migration guide; proxy deferred to Phase 2 to reduce complexity.
- **Security gaps:** basic invariants + Slither; external audit planned for Phase 2.

---

12) Governance, licensing & transparency

- Public repos; **MIT/Apache-2.0** for code; **CC BY 4.0** for docs.
- Lightweight changelog; parameters adjustable by multisig (Phase 1), timelock/governance deferred to Phase 2.
- **Biweekly** status notes during the 12-week window.

---

13) Team & roles (lean)

- **Tech Lead / Solidity** — contracts & reviews.
- **Frontend/SDK** — UI alpha-lite and TS client.
- **QA/Sec (part-time)** — tests, invariants, Slither.
- **DevOps (part-time)** — CI, deployments, testnet ops.
- **PM (part-time)** — planning, coordination, reporting.

---

14) Dependencies & assumptions

- EVM-compatible environment; kBTC token interface; testnet faucet or provisioning.
- (Optional) external ERC-20 with `MINTER_ROLE` granted to the adapter.
- Community feedback during public testnet.

---

15) Next steps (Phase 2 candidates)

- Add **slashing**, **risk oracles**, **dynamic reward curves**, and **timelocked governance**.
- Formal verification/audit and richer dashboards/subgraph.
- Portability work toward **PolkaVM**.

Lean, practical, and shippable in ~12 weeks with a $25k budget.