TIMLG Protocol: A Decentralized Causality & Prediction Experiment using the NIST Randomness Beacon [*]

Richard David Martín Martín
support@timlg.org
https://www.timlg.org

Version v1.2 — 2026-02-22

Abstract

TIMLG Protocol is a public, auditable commit–reveal experiment in which participants attempt to predict future bits derived from the NIST Randomness Beacon. Each participation is a *ticket* that escrows exactly 1 TIMLG and settles deterministically: WIN → stake returned + reward minted (net of protocol fee), LOSE → stake burned, NO-REVEAL/invalid → stake burned. The protocol is designed to test whether any strategy can exceed chance performance under strict anti-leakage constraints (the "Hawking Wall"). This version (v1.2) incorporates the *Live Audit Infrastructure* and the *Audit Dashboard* as core evidential components for real-time transparency.

***K*eywords** commit–reveal, randomness beacon, NIST, oracle, Solana, tokenomics, hypothesis testing

1 Motivation and Non-Claims

1.1 What this document is

This document describes an *experiment*, not an investment product. It specifies a public, auditable protocol that measures prediction performance against a pre-registered target derived from the NIST Randomness Beacon.

1.2 What the protocol measures

The protocol measures whether any participant strategy can predict future beacon-derived bits above chance *under strict anti-leakage constraints* (the "Hawking Wall").

1.3 What the protocol does not claim

No claims of "time messaging" or guaranteed profit are made. The protocol does not claim that beating chance is possible, only that it can be *measured* under constraints. Any anomaly must be replicated and must rule out mundane explanations (bias, leakage, manipulation) before interpretation.

1.4 Why on-chain

The blockchain serves as the immutable audit log: anyone can verify round parameters, commitments, finalization evidence, and settlement outcomes.

[*] *Status*: Beta specification for implementation and testing. No profit guarantees; experimental protocol.

1.5 How to read the rest of the paper

Section 2 states the hypothesis framework (H0–H5). Sections 3–10 specify goals, protocol mechanics, BitIndex/Treasury, oracles, security, and implementation notes. Sections 11–12 cover roadmap and risk disclosures.

2 Hypotheses (H0–H5)

TIMLG interprets outcomes using a simple hypothesis ladder. The purpose is to *pre-register* how results should be read, and to ensure that apparent advantages are first explained by ordinary causes before any stronger interpretation is considered.

2.1 H0: Pure chance (null hypothesis)

Participants have no predictive advantage. Under H0, each ticket is a Bernoulli trial with $p = 0.5$ and observed deviations are attributed to statistical fluctuation. The default expectation is that, over large samples, performance converges to chance.

2.2 H1: Bias / implementation artifact

Apparent advantage is explained by errors or bias in the protocol pipeline, such as: bit-extraction conventions (indexing/order), serialization mismatches, incorrect round targeting, client/UI bugs, or settlement edge-cases. Under H1, an "edge" may appear only for specific bit positions, time ranges, or software versions.

2.3 H2: Operational leakage

Apparent advantage is explained by information leakage or timing errors: commits made after the target is partially knowable, side-channels in round scheduling, or off-chain coordination that exploits leaked data. Under H2, tightening timing constraints and verification procedures should remove the edge.

2.4 H3: Oracle / relayer manipulation

Apparent advantage is explained by adversarial infrastructure: oracle equivocation, censorship, delayed finalization, conflicting pulse reports, or relayer behavior that selectively includes/excludes commits or reveals. Under H3, strengthening oracle quorum/evidence rules and making infrastructure attacks observable should remove the edge.

2.5 H4: Non-trivial correlations

After H1–H3 are ruled out, residual advantage may indicate a reproducible correlation between the target bits and some accessible signal that does not constitute direct leakage. H4 is framed operationally: the correlation must be measurable, replicable, and stable under stricter controls.

2.6 H5: Exotic interpretive framing (last resort)

Only after H1–H4 are systematically excluded, remaining anomalies may be discussed under more speculative interpretations. Importantly, H5 is not a claim: it is a reminder that interpretations are downstream of replication and elimination of mundane explanations.

2.7 Operational implication

This ladder is an *escalation policy*: when anomalies appear, the protocol responds by tightening constraints and increasing scrutiny rather than by relaxing rules or increasing rewards.

3 Design Goals

This section states the protocol goals as engineering requirements. They are written to prioritize auditability, falsifiability, and robustness over feature breadth.

3.1 Primary goals

- **Falsifiability:** outcomes must be testable against a clear baseline (chance) with pre-registered interpretation rules.
- **Public verifiability:** any third party should be able to replay the protocol rules and verify round targets, eligibility, and settlement from public data.
- **Anti-leakage by design (Hawking Wall):** the protocol must minimize opportunities for timing leaks, post-hoc adaptation, and selective disclosure.

3.2 Protocol integrity goals

- **Deterministic settlement:** ticket outcomes depend only on committed inputs, the finalized target, and fixed rules; no discretionary intervention.
- **Commit–reveal correctness:** commits must be binding and reveals must be verifiable, with clear handling of invalid reveals and missed windows.
- **Abort resistance:** the protocol must make “not revealing” strictly worse than revealing, preventing selective outcome reporting as a strategy.

3.3 Economic design goals

- **Uniform sample weight:** each ticket represents the same unit of stake and the same unit of measurement.
- **Minimal monetary policy complexity:** token flows should be simple enough to audit and reason about under H0, while remaining robust to adversarial behavior.

3.4 Scalability and usability goals

- **Low marginal participation cost:** support batching and relayed submission so that the experiment can collect large sample sizes without prohibitive fees.
- **State efficiency:** on-chain state should grow predictably and avoid per-ticket storage where practical (e.g., through batching/commitment aggregation in later phases).

3.5 Evolution goals

- **Escalation over hype:** if anomalies appear, the protocol should tighten constraints, increase evidence requirements, and favor replication over interpretation.
- **Upgradeable early, lockable later:** during prototyping, upgrades may be required; the long-term goal is minimizing trusted upgrade paths.

4 System Overview

This section provides a high-level map of the system: who the actors are and how a round progresses. Detailed rules are specified in later sections.

4.1 Actors

- **Participants (“Participants”):** submit predictions as tickets using a commit–reveal workflow.
- **Relayer(s):** optionally batch and submit user-signed messages on-chain to reduce participation friction and fees. Relayers are not trusted for correctness.
- **Messengers / Oracles:** fetch the target beacon pulse and publish it on-chain with signature verification (using the implemented Ed25519 introspection).
- **Live Audit Service:** an automated verifier that provides categorical proof of token flow, integrity proofs, and deterministic settlement validation.
- **On-chain programs:** enforce state transitions, validate commits/reveals, verify oracle signatures, and settle outcomes (manually or via automated supervisor).
- **Treasury Vaults:** isolated accounts for protocol fees (`reward_fee_pool`) and SOL service fees (`treasury_sol`), funding operations and infrastructure.

4.2 Core objects

- **Round:** a time-bounded unit of the experiment that targets a specific future beacon pulse and defines commit and reveal windows.
- **Ticket:** a single participation instance within a round, with a fixed stake and a binding commit.
- **Target Pulse:** the round's authoritative pulse output published by the oracle and used for settlement.

4.3 Round lifecycle (v0.1)

A round follows a strict sequence:

1. **Round definition:** a future target pulse index is selected and parameters are set on-chain.
2. **Commit window:** participants submit ticket commitments and escrow the fixed stake.
3. **Pulse publication:** after the target pulse is available, the oracle sets the pulse on-chain (*pulse_set* status), enabling the reveal window.
4. **Reveal window:** participants reveal their committed guess and salt; the program verifies and records results.
5. **Settlement and Sweep:** the round is finalized; tokens are settled (mint/burn); residual funds may be swept to the treasury after a grace period.

Pipelined scheduling. To keep the experiment continuous, the operator typically maintains a *pipeline* of several future rounds. Each round stores its own `commit_deadline_slot` and `reveal_deadline_slot` on-chain. On the current Devnet deployment, the operator enforces a maximum betting horizon of roughly 420 s (about 1050 slots at ~400 ms/slot) to avoid creating rounds whose commit window is too far in the future.

4.4 Design separation

For clarity, this paper separates:

- **Targets** (how the beacon pulse and bits are defined) in Section 5,
- **Mechanics** (tickets, commit–reveal, settlement) in Section 6,
- **BitIndex and Treasury policy** in Section 7,
- **Oracle rules** in Section 8,
- **Threat model** in Section 9.

5 Targets: NIST Beacon, Pulse Selection, Bit Extraction

This section defines the measurement target: which beacon pulse is used for a round and how a single bit is derived from that pulse in a deterministic, testable way.

5.1 Beacon source

TIMLG targets bits derived from the NIST Randomness Beacon. A beacon *pulse* is identified by an index (or equivalent identifier) and provides a public output value that can be independently retrieved and verified off-chain.

5.2 Pulse selection (anti-leakage requirement)

For each round, the protocol targets a *future* beacon pulse. The pulse identifier must be fixed and publicly known *before* the commit window closes. This prevents strategies that depend on already-published outputs and reduces timing leakage risk.

Concretely, a round defines:

- a target pulse identifier (e.g., an integer pulse index),
- a commit window that closes strictly before the pulse becomes knowable,
- a reveal window that opens after the pulse is published on-chain (*pulse_set*).

In the current Devnet operator, rounds are *pipelined*: multiple future targets are created in advance so that users always have an open commit window. The operator enforces a maximum betting horizon of roughly 420 seconds (see Appendix A), so if the next available target would close too far in the future, the pipeline waits rather than creating excessively long commit windows.

5.3 Canonical output representation

To avoid ambiguity across implementations, the protocol defines a canonical representation of the pulse output used for bit extraction:

- the output value is treated as a fixed-length 512-bit string (64 bytes),
- the bit indexing convention follows a little-endian (LSB-first) mapping: bit 0 is the LSB of byte 0.

Any implementation must follow the same convention to ensure identical settlement results.

5.4 Bit extraction

Given a pulse P (64 bytes) and a bit index $i \in [0, 511]$, the target bit b is extracted as:

- $byte\_i = \lfloor i/8 \rfloor$
- $bit\_i = i \pmod 8$
- $b = (P[byte\_i] \gg bit\_i) \,\&\, 1$

This ensures a deterministic and portable mapping from pulse bytes to prediction targets.

5.5 Bit index assignment (overview)

Each ticket is associated with a bit index i for extraction. The protocol uses a deterministic assignment rule designed to prevent participants from selecting or grinding favorable indices and to avoid crowding on a single bit. The exact assignment rule is specified in Section 7.

5.6 Consistency and test vectors

This protocol requires unambiguous test vectors:

- at least one published pulse output and the corresponding extracted bits under the canonical convention,
- reference examples for $bitIndex$ computation (when applicable),
- a conformance check that independent implementations produce identical b for the same round and ticket inputs.

These vectors act as the “ground truth” for client and oracle correctness.

6 Protocol Mechanics: Tickets, Commit–Reveal, Settlement

This section specifies the mechanics of participation: how a ticket is created, how a commit binds a prediction, how a reveal proves it, and how settlement is computed.

6.1 Tickets

A **ticket** is a single participation instance within a round. Each ticket:

- escrows a fixed stake amount (1 TIMLG),
- contains exactly one binary guess (0 or 1),
- is bound by a cryptographic commitment submitted during the commit window,
- may be revealed only during the reveal window.

Tickets are independent samples for statistical measurement.

6.2 Commit phase (binding)

During the commit window, the participant submits a commitment that binds their guess without revealing it. Each ticket uses a fresh secret salt and a per-ticket nonce.

Let:

- $roundId$ be the round identifier,
- pk be the participant public key,
- $g \in \{0, 1\}$ be the guess bit,
- s be a secret salt,
- n be a ticket nonce (replay protection / uniqueness).

The ticket commitment is:

$$C = \text{SHA256}(\text{"commit"} \| roundId_{le} \| pk \| nonce_{le} \| g \| salt_{32})$$

where || denotes concatenation and le denotes little-endian encoding. On-chain, a valid commit:

- records C for the ticket,
- escrows exactly 1 TIMLG into a program-controlled stake vault,
- collects a fixed **SOL service fee** (lamports) used for network and operational costs,
- enforces window validity (must be within the commit window).

6.3 Reveal phase (verification)

During the reveal window, the participant reveals (g, s_{32}, n). The program recomputes:

$$C' = \text{SHA256}(\text{"commit"} \| roundId_{le} \| pk \| n_{le} \| g \| s_{32})$$

and accepts the reveal if and only if $C' = C$.

A reveal is considered **invalid** if it fails verification, is duplicated, or is submitted outside the reveal window.

6.4 Outcome determination

For a ticket with a valid reveal, the target bit b is compared against the guess g.

- **WIN:** valid reveal and $g = b$.
- **LOSE:** valid reveal and $g \neq b$.
- **NO-REVEAL:** no valid reveal submitted (treated as burn).

6.5 Settlement (token flows)

Settlement applies the following rules:

- **WIN:** return the 1 TIMLG stake and mint a reward of +1 TIMLG. A protocol fee (`reward_fee_bps`) is applied to the reward before minting; on the current Devnet deployment the fee is set to **100 bps (1%)**.
- **LOSE:** burn the 1 TIMLG stake. This action is irreversible and contributes to the protocol's self-balancing supply model.
- **NO-REVEAL:** burn the 1 TIMLG stake (same outcome as LOSE).

6.6 Claim and Sweep

Winners must claim their rewards; unclaimed funds remain in the round's vault. After a *claim grace period*, an operator may perform a **sweep** to transfer residual SOL and SPL tokens to the protocol treasuries and close the round.

This separation ensures that non-reveals cannot be used to selectively report only favorable outcomes, and that accounting remains auditable.

6.7 Batching and relayed submission (overview)

Participants may submit commits and reveals directly or via relayers that batch transactions. Relayers improve usability and cost efficiency but do not change correctness: the on-chain program validates all messages and window constraints deterministically. Implementation details for batching are provided in Section 10.

7 BitIndex and Treasury Policy

This section specifies (i) how each ticket is assigned a bit position (`bitIndex`) and (ii) how the Treasury Vault is used as a protocol integrity mechanism.

7.1 Why per-ticket BitIndex

Using a single global bit position for all participants would concentrate outcomes and incentives on one bit, increasing variance and making the experiment more sensitive to coordinated behavior. TIMLG assigns a **bit index per ticket** to:

- reduce "crowding" on a single bit,
- prevent participants from selecting or grinding favorable indices,
- preserve measurability while spreading exposure across the 512-bit output space.

7.2 Deterministic BitIndex assignment

The bit index $i \in \{0, \ldots, 511\}$ is assigned deterministically from the round and ticket public data:

$$i = \text{u16\_le}(\ \text{SHA256}(\ \text{"bitindex"} \| roundId_{le} \| pk \| nonce_{le}\ )[0..2]\ ) \bmod 512$$

where $\text{u16\_le}(\cdot)[0..2]$ indicates interpreting the first two bytes of the hash as a little-endian unsigned 16-bit integer.

7.3 Canonical assignment requirement

The assignment must be independent of the round's target pulse to avoid adaptive selection. By using the participant's public key and a per-ticket nonce, the protocol ensures that different tickets from the same user are likely to target different bits, providing better statistical coverage.

7.4 Treasury Vault purpose

The Treasury Vault is a protocol sustainability mechanism. Its purpose is to:

- receive protocol fees from winning rewards (`reward_fee_pool`) in TIMLG SPL tokens,
- receive **SOL service fees** and residual tokens from round sweeps,
- fund protocol operations (relaying, oracle operations, maintenance),
- support the **Live Audit Infrastructure** and public evidence hosting.

7.5 No-reveal policy (integrity rule)

A ticket that does not produce a valid reveal within the reveal window is resolved as **NO-REVEAL**. For NO-REVEAL tickets:

- the escrowed 1 TIMLG stake is **burned**,
- no reward is minted.

This rule ensures that participants cannot selectively report only favorable outcomes, as hiding a result is economically equivalent to a loss. This rule removes selective disclosure incentives: it is never advantageous to hide an outcome by failing to reveal. All tokens involved are either minted or burned deterministically.

7.6 Automated Cleanup (Residue Zero)

The protocol provides an automated cleanup mechanism where finished tickets can be closed to recover the on-chain rent deposit. This ensures that the system maintains a "residue zero" footprint for active participants.

7.7 Treasury use policy (high-level)

Treasury funds may be used only for protocol sustainability and experiment integrity, such as:

- relayer operations and transaction sponsorship,
- oracle/messenger operations and evidence publication costs,
- security reviews, audits, and bug bounties,
- implementation and maintenance under governance-approved processes.

Detailed governance, reporting, and spending controls are specified elsewhere (or in a later version).

8 Oracle / Messenger Layer

This section specifies how the protocol obtains an authoritative target pulse output for each round. The oracle/messenger layer provides a bridge from the external beacon to on-chain finalization with public evidence.

8.1 Purpose

On-chain programs cannot directly fetch external data. Therefore, TIMLG relies on a messenger/oracle process that:

- retrieves the target beacon pulse for the round,
- publishes the pulse output on-chain via a signed message,
- enables the on-chain program to verify the signature and set the pulse for settlement.

8.2 Single Oracle (v0.1)

In the initial version (v0.1), the protocol relies on a configured **Oracle Public Key**. The on-chain program accepts a pulse only if it is accompanied by a valid Ed25519 signature from this specific key.

8.3 Signed Pulse Message

The oracle submits a message with the following structure:

$$M_{pulse} = \text{prefix} \| programID \| roundId_{le} \| pulseIndex_{le} \| pulse_{64}$$

The program uses Ed25519 instruction introspection to verify that the message was signed by the authorized oracle and matches the round's requirements.

8.4 Operational Transparency

While v0.1 uses a single oracle, all actions are recorded on-chain. Third parties can independently fetch the pulse from the NIST beacon and verify that the on-chain pulse matches the official source.

8.5 Roadmap: Quorum and Evidence

Moving toward greater decentralization, future versions will introduce:

- **Oracle Quorum:** multiple independent operators must attest to the same pulse.
- **On-chain Evidence:** storage of hashes or references to publicly verifiable evidence logs.

This hardening will further reduce reliance on a single operator.

9 Security and Threat Model

This section summarizes the threat model and the protocol's security posture. The goal is not to claim perfect security, but to define what is defended, how, and what risks remain.

9.1 Security objectives

- **Integrity of measurement:** outcomes must reflect the protocol rules and a fixed target, not post-hoc adaptation or selective disclosure.
- **Target correctness:** the finalized beacon output used for settlement must be unambiguous and publicly auditable.
- **Censorship awareness:** if infrastructure actors censor or delay messages, this should be observable and should not silently bias results.

9.2 Threats and mitigations

T1. Timing leakage (commit after target is knowable). *Threat:* Participants gain an edge by committing after the target pulse becomes available or partially inferable.
Mitigation: rounds target a future pulse fixed before commit close; strict commit windows enforce ordering.
Residual risk: clock drift, network latency, and misconfigured windows can reduce safety margins.

T2. Selective disclosure (abort / non-reveal strategy). *Threat:* Participants reveal only when favorable, biasing measured performance.
Mitigation: NO-REVEAL/invalid reveals result in the stake being **burned** (economically equivalent to a loss and strictly worse than a win).
Residual risk: participants may still accept forfeiture if external incentives dominate; the protocol treats this as data (dropout) and not as a valid win.

T3. BitIndex grinding / crowding. *Threat:* Participants attempt to choose favorable bit positions or concentrate behavior on a single bit.
Mitigation: deterministic per-ticket BitIndex assignment derived from SHA256 of the round and ticket metadata; participants cannot select i.
Residual risk: without a future external seed, a participant with massive compute could theoretically grind nonces to find a slightly better bit index, though the cost of doing so is expected to far exceed any marginal gain.

T4. Oracle equivocation or incorrect reporting. *Threat:* Oracle publishes a wrong pulse or fails to report.
Mitigation: single oracle public key with mandatory Ed25519 signature verification; pulse must match the expected round and target index.
Residual risk: reliance on a single operator; this is mitigated by public auditability of the NIST beacon and will be further addressed by the roadmap's quorum and evidence requirements.

T5. Relayer censorship or message withholding. *Threat:* A relayer selectively includes commits/reveals, delays inclusion, or refuses service.
Mitigation: direct submission remains possible; support multiple relayers; window rules make late submission invalid regardless of relayer intent.
Residual risk: users relying on a single relayer may experience denial of service; operational redundancy is recommended.

T6. Replay and message forgery. *Threat:* Attackers replay old commits/reveals or forge messages for other users.
Mitigation: signature verification, domain separation, per-ticket nonces, and strict round binding in the commit hash.
Residual risk: client key compromise and user operational security failures are out of scope.

T7. Denial of service and state bloat. *Threat:* Spammers attempt to overload the system or grow on-chain state without bound.
Mitigation: fixed stake per ticket introduces an economic cost; batching reduces overhead per ticket.
Residual risk: sufficiently funded adversaries can still attempt congestion; resource limits and protocol parameters may need adjustment.

T8. Sybil volume (statistical farming). *Threat:* An adversary uses many identities/tickets to dominate incentives or produce extreme streaks by chance.
Mitigation: economic cost per ticket; incentive design should avoid rewarding extreme outliers without anti-sybil controls.
Residual risk: sybil resistance is not solved in general; it is managed via economics and careful incentives.

9.3 Disclosure

The protocol should be treated as an evolving system. New threats may be discovered as usage grows; the intended response to anomalies is escalation (tighter windows, stronger oracle rules, audits), not denial.

10 Implementation Notes (Solana)

This section provides implementation notes for deploying TIMLG on Solana. It is not a full specification of account layouts or instruction binary formats; those are intended for a separate implementation specification once parameters are finalized.

10.1 Why Solana

Solana is selected for high-throughput, low-latency execution and low marginal transaction cost, enabling large sample sizes. The design targets a user experience where participants can submit signed messages with minimal fee exposure, while settlement remains fully verifiable on-chain.

10.2 Program architecture

A minimal deployment can be implemented as one program, but a modular split is also possible. The design assumes the following on-chain responsibilities:

- round creation and state transitions (commit window, reveal window, finalized target state),
- commit and reveal validation,
- deterministic settlement and accounting updates,
- treasury vault custody and controlled spending interface.

10.3 Core accounts (high-level)

An implementation typically uses Program Derived Addresses (PDAs) for deterministic state:

- **Config PDA:** global parameters (window lengths, oracle_pubkey, treasury pubkeys, fee basis points).
- **Round PDA:** per-round state (target pulse index, window slots, *pulse_set* status, pulse value, finalized/swept flags).
- **Round Vaults:** each round controls a SOL *vault* (for lamports) and an SPL *timlg_vault* (for tokens).
- **Treasury PDAs:** protocol-wide `treasury_sol` (lamports), and SPL vaults such as `reward_pool` and `replication_pool`.

Per-ticket storage is handled via Ticket PDAs containing the commitment and outcome state.

10.4 Token model (SPL)

TIMLG is implemented as an SPL token. The protocol assumes:

- a fixed stake amount per ticket (1 unit),
- program-controlled vault accounts for escrow and treasury,
- mint/burn permissions restricted to the protocol program for deterministic settlement.

Mint/burn paths must be fully auditable and solely triggered by the settlement rules.

10.5 Relayed submission and batching

To reduce user friction, commits and reveals may be submitted by relayers. Participants sign messages off-chain; a relayer aggregates them into transactions that call program instructions.

The protocol remains trust-minimized because:

- the program verifies signatures and round bindings for all messages,
- window constraints are enforced on-chain,
- relayers cannot fabricate valid commits/reveals for users without their signatures.

Batching is an engineering optimization. Two common approaches are:

- **Batch instructions:** submit arrays of commits/reveals in a single transaction, verifying each.
- **Commitment aggregation:** store a compact commitment (e.g., Merkle root) on-chain and provide inclusion proofs at reveal/settlement time (roadmap optimization).

10.6 Compute and state considerations

Solana programs are constrained by compute budgets and account sizes. Therefore:

- early versions may prefer smaller batches for reliability,
- state growth should be bounded or prunable over time,
- later phases should prioritize compact commitments and minimized per-ticket state.

These constraints motivate an incremental approach: correctness first, then optimization.

10.7 Oracle integration on Solana

Oracle pulses are submitted as on-chain instructions (`set_pulse_signed`) that update the Round PDA. The program verifies:

- the signature via **Ed25519 program introspection**, ensuring the signature covers the round ID, pulse index, and pulse data,
- that the signer matches an authorized oracle public key from the on-chain oracle set,
- that the signed message correctly references the round's target pulse index and domain separator.

Once set, the pulse becomes immutable for the round, and the reveal window is considered open.

10.8 Upgrades and deployment posture

Early versions may require upgrades to fix bugs and adjust parameters. The protocol should therefore:

- explicitly disclose the upgrade authority model during bootstrapping,
- aim to minimize privileged actions over time,
- adopt audits and change control before mainnet hardening.

10.9 Implementation deliverables

A credible implementation plan should include:

- conformance tests for canonical bit extraction and ticket verification,
- reference clients for commit/reveal signing,
- reproducible builds and public verification scripts for rounds and outcomes.

11 Roadmap

This roadmap prioritizes verifiable deliverables. Each phase is considered complete only when its artifacts can be independently reproduced and audited.

11.1 Phase 1: Multi-Round Core & Devnet Operations

- **Status: Operational.** End-to-end lifecycle (Commit → Reveal → Settlement) is fully implemented and verified on Devnet.
- Implementation of deterministic tokenomics (mint/burn) and auto-supervisor for round transitions.

11.2 Phase 2: Public Audit Infrastructure (Live Audit)

- **Status: Integrated.** Deployment of real-time Audit Dashboards and integrity proof generation.
- Categorical proof of token flow and categorical validation of oracle randomness impact.

11.3 Phase 3: Oracle Quorum & Scale Testing

- **Status: Integrated.** Support for decentralized oracle sets and Ed25519 batch verification.
- Hardening of pulse publication timelines and 24/7 autonomous operation benchmarks.

11.4 Phase 4: Decentralized Relayers & UX Hardening

- **Status: In Progress.** Development of public relayer infrastructure to enable gasless user participation.
- Optimization of compute usage through batched commit/reveal arrays and state pruning.
- Integration of premium non-custodial interfaces for high-frequency interaction.

11.5 Phase 5: Mainnet Alpha & Security Maturation

- Complete at least one independent security review and address findings publicly.
- Launch with conservative parameters (windows, quorum) and multisig governance.
- Establish transparent treasury reporting and operational budgets.

11.6 Ongoing: Replication and anomaly response

- Periodically publish statistical reports and methodology updates.
- If anomalies appear, apply the escalation policy: tighten constraints, increase evidence requirements, and prioritize replication over interpretation.

12 Risks and Disclosures

This section discloses risks and clarifies scope. It is intentionally explicit.

12.1 Not an investment product

TIMLG is an experimental protocol designed to measure prediction performance under strict constraints. It is not an investment product, not a promise of profit, and not financial advice. Participation may result in partial or total loss of tokens.

12.2 Outcome variance and expected loss scenarios

Each ticket is a binary trial with high variance. Under the null hypothesis (chance performance), outcomes will fluctuate and streaks may occur naturally. Participants should assume:

- losses are expected and common,
- short-term results are not evidence of an edge,
- "winning streaks" can occur by chance without implying predictability.

12.3 Operational risks

Users may lose funds due to operational factors, including but not limited to: Users may lose funds due to operational factors, including but not limited to:

- missed reveal windows (NO-REVEAL results in stake **burn**),
- protocol fees deducted from winning rewards (`reward_fee_bps`),
- sweeping of unclaimed funds after the claim grace period,
- client-side mistakes (incorrect signing, reused salts, wrong nonces),

- network congestion or service interruptions,
- reliance on third-party relayers or front-ends.

12.4 Refund conditions

If the oracle fails to publish a pulse, the protocol provides a **refund** mechanism after a timeout (`REFUND_TIMEOUT_SLOTS`). This is a slot-based condition. Once the timeout passes, anyone can trigger the recovery of funds to return escrowed stakes to participants.

12.5 Smart contract and infrastructure risk

Smart contracts may contain bugs. Oracle/messenger infrastructure may fail or behave adversarially. While the protocol is designed for auditability and escalation, these risks cannot be eliminated. Users should treat early versions as experimental and assume the possibility of faults.

12.6 No exotic claims

The protocol makes no claim of "time messaging," retrocausal signaling, or any exotic physical mechanism. If anomalies are observed, the default interpretation is that they require replication and the elimination of mundane explanations (bias, leakage, manipulation) before any further discussion.

12.7 Regulatory and legal uncertainty

Token-based systems may be subject to regulatory requirements that vary by jurisdiction and may change over time. Users are responsible for understanding and complying with applicable laws, tax obligations, and exchange restrictions.

12.8 No warranties

The protocol and associated software are provided "as is" without warranties of any kind. Use at your own risk.

A Parameters

Parameter	Value
bitCount	512
stake	1 TIMLG (configurable; Devnet currently 1)
token decimals	9 (SPL standard)
bitIndex mode	per-ticket deterministic (derived from commit data)
commit window	*Per-round* deadline in slots; operator caps **betting time** to **~420s** (~1050 slots at 400ms/slot)
reveal window	**1000 slots** (Devnet as of 2026-02-02; ~6m40s)
claim grace ("refund")	150 slots (Devnet default; configurable)
reward fee	**0 bps** (Devnet as of 2026-02-02; configurable)
oracle	Single operator pubkey verification (Devnet uses one oracle key)

B Message Schemas

To ensure domain separation and prevent replay attacks, the protocol defines exact binary schemas for messages that are signed off-chain and verified on-chain. All multi-byte fields are little-endian unless otherwise specified.

Commit Message (`commit_v1`)

1. **Prefix:** `"timlg-protocol:commit_v1"` (ASCII)
2. **Program ID:** 32 bytes
3. **Round ID:** 8 bytes (u64, little-endian)
4. **User Pubkey:** 32 bytes
5. **Nonce:** 8 bytes (u64, little-endian)

6. **Commitment:** 32 bytes (SHA256)

Reveal Message (`reveal_v1`)

1. **Prefix:** `"timlg-protocol:reveal_v1"` (ASCII)
2. **Program ID:** 32 bytes
3. **Round ID:** 8 bytes (u64, little-endian)
4. **User Pubkey:** 32 bytes
5. **Nonce:** 8 bytes (u64, little-endian)
6. **Guess:** 1 byte (0 or 1)
7. **Salt:** 32 bytes

Pulse Message (`pulse_v1`)

1. **Prefix:** `"timlg-protocol:pulse_v1"` (ASCII)
2. **Program ID:** 32 bytes
3. **Round ID:** 8 bytes (u64, little-endian)
4. **Pulse Index:** 8 bytes (u64, little-endian)
5. **Pulse Data:** 64 bytes (512 bits)

References