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

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

Version v0.1 (Draft) — 2025-12-28

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 → payout 2, LOSE → payout 0, NO-REVEAL/invalid → forfeiture to a Treasury Vault. The protocol is designed to test whether any strategy can exceed chance performance under strict anti-leakage constraints (the "Hawking Wall").

***K*eywords** commit–reveal, randomness beacon, NIST, oracle quorum, 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.

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.

* *Status*: Draft specification for discussion and implementation. No profit guarantees; experimental protocol.

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, provide attestations and evidence, and finalize the official target under a quorum rule.
- **On-chain programs:** enforce state transitions, validate commits/reveals, finalize targets, and settle outcomes deterministically.
- **Treasury Vault:** receives penalties from missed/invalid reveals and funds protocol operations according to policy.

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.
- **Finalized target:** the round's authoritative pulse output (or its hash) agreed by the oracle quorum and used for settlement.

4.3 Round lifecycle (high-level)

A round follows a strict sequence:

1. **Round definition:** a future target pulse is selected and the round parameters are made public.
2. **Commit window:** participants submit ticket commitments and escrow the fixed stake.
3. **Finalization:** after the target pulse is available, messengers/oracles attest to the pulse and a quorum finalizes the official target.
4. **Reveal window:** participants reveal their committed guess and secret to enable verification.
5. **Settlement:** each ticket is resolved deterministically (WIN/LOSE/NO-REVEAL) and accounting is updated.

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 only after the target is finalized.

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,
- the encoding used for hashing and indexing (hex/base64, endianness) is fixed by specification,
- the bit indexing convention is fixed: $bitIndex \in \{0, \ldots, 511\}$.

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

5.4 Bit extraction

Given a finalized pulse output O (512 bits) and a bit index $i \in [0, 511]$, the target bit is:

$$b = \text{bit}(O, i)$$

where $\text{bit}(O, i)$ returns the i-th bit under the protocol's fixed indexing convention.

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:

$$C = H(\, roundId \| pk \| g \| s \| n \,)$$

where $H(\cdot)$ is a domain-separated cryptographic hash specified by the implementation.

On-chain, a valid commit:

- records C for the ticket,
- escrows exactly 1 TIMLG into a program-controlled stake vault,
- enforces window validity (must be within the commit window).

6.3 Reveal phase (verification)

During the reveal window, the participant reveals (g, s, n) for the ticket. The program recomputes:

$$C' = H(\, roundId \| pk \| g \| s \| n \,)$$

and accepts the reveal if and only if $C' = C$ for the recorded commitment and the reveal is submitted within the reveal window.

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 protocol compares the revealed guess g against the ticket's target bit b (defined in Section 5, with the ticket's $bitIndex$ defined in Section 7).

- **WIN:** valid reveal and $g = b$.
- **LOSE:** valid reveal and $g \neq b$.
- **NO-REVEAL:** no valid reveal submitted within the reveal window (including invalid reveals).

6.5 Settlement (token flows)

Settlement is deterministic and applies exactly one of the following rules:

- **WIN:** return the escrowed stake and mint a reward of +1 TIMLG to the participant (total payout = 2).
- **LOSE:** burn the escrowed stake (total payout = 0).
- **NO-REVEAL:** transfer the escrowed stake to the Treasury Vault (no mint and no burn).

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

6.6 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

Each ticket is assigned a bit index $i \in \{0, \ldots, 511\}$ deterministically from public inputs. The assignment must be:

- **non-interactive:** the participant cannot choose i,
- **replay-safe:** different tickets produce different indices with high probability,
- **publicly verifiable:** any third party can recompute i from the round and ticket data.

Let:

- $seed$ be a public seed fixed before commit close (e.g., derived from a previous finalized pulse),
- $roundId$ be the round identifier,
- pk be the participant public key,
- n be the ticket nonce.

The bit index is:

$$i = H(\, seed \| roundId \| pk \| n \,) \bmod 512$$

where $H(\cdot)$ is a domain-separated hash defined by the protocol.

7.3 Canonical seed requirement

The seed must be determined *before* the commit window closes and must be independent of the round's target pulse. This avoids adaptive selection of i based on future target data. The protocol specifies the exact seed derivation in the parameterization (Appendix A).

7.4 Treasury Vault purpose

The Treasury Vault is not a "house" that pays winners. Its purpose is to:

- absorb stake forfeitures from NO-REVEAL/invalid reveals,
- fund protocol operations (relaying, oracle operations, audits) under an explicit policy,
- reduce supply noise by avoiding mint/burn on the NO-REVEAL path.

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 transferred to the Treasury Vault,
- no reward is minted and no stake is burned for that ticket.

This rule removes selective disclosure incentives: it is never advantageous to hide an outcome by failing to reveal.

7.6 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,
- provides attestations that reference verifiable evidence,
- enables the on-chain program to finalize a single canonical target for settlement.

8.2 Attestations

For a given round and its target pulse identifier, a messenger/oracle submits an attestation containing:

- the round identifier and target pulse identifier,
- the pulse output (or a commitment/hash to it under the canonical representation),
- an evidence reference (e.g., a content-addressed object and its hash).

The protocol fixes the canonical encoding for output hashing and the evidence format in appendices or implementation parameters.

8.3 Evidence requirement

Each attestation must reference evidence sufficient for independent verification that the attested output corresponds to the target pulse. Evidence should be publicly retrievable and immutable (or content-addressed), so third parties can audit oracle correctness.

8.4 Quorum and finalization (conceptual)

A round is finalized when a quorum of oracle members attest to the same canonical output commitment. Once finalized:

- the canonical output commitment (and/or output value) becomes immutable round state,
- settlement for all tickets in the round uses this finalized target.

Quorum size, membership, and voting rules are parameters that may evolve; the core requirement is that the finalized target is unambiguous and publicly auditable.

8.5 Finalized target interface

From the perspective of ticket settlement, the oracle layer outputs exactly one value per round:

- a finalized pulse output commitment (and optionally the output itself) under the canonical representation.

This value is the sole external-data dependency for outcome determination.

8.6 Operational notes

The protocol may support multiple independent messengers/oracles and multiple relayers. Correctness does not depend on trusting a single operator; instead, correctness is enforced by quorum agreement and public evidence.

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 forfeit stake to the Treasury Vault (strictly worse than revealing).
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 from a public seed; participants cannot select i.
Residual risk: poor seed design or ambiguous encoding could introduce bias (addressed via canonicalization and test vectors).

T4. Oracle equivocation or incorrect reporting. *Threat:* Oracles attest conflicting outputs, report wrong pulses, or finalize an incorrect target.
Mitigation: quorum-based finalization and evidence requirements; third-party auditability of evidence.
Residual risk: collusion among quorum members; this is reduced by decentralizing membership and by strengthening quorum/verification over time.

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, quorum parameters, canonical encoding version).
- **Round PDA:** per-round state (target pulse identifier, window timestamps, finalized target commitment).
- **Stake Vault:** SPL token account holding escrowed stakes during a round.
- **Treasury Vault:** SPL token account receiving NO-REVEAL forfeitures.
- **Oracle Set / Membership:** oracle registry and quorum configuration (exact layout is implementation-defined).

Per-ticket storage can be implemented directly (ticket accounts) for early versions, and later optimized via batching and commitments (see below).

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 attestations are submitted as on-chain instructions that update round state toward finalization. The program should store:

- the canonical target commitment for the finalized pulse output,
- sufficient metadata to audit which oracle members attested and how quorum was reached.

Evidence references can be stored as hashes/identifiers to keep on-chain state small while preserving verifiability.

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 0: Specification freeze (v0.1 → v0.2)

- Finalize canonical encoding rules (pulse output representation, bit indexing convention).
- Publish conformance test vectors for bit extraction and ticket verification.
- Freeze message schemas for commit/reveal/attestations (domain separation, nonces, expiry).
- Define initial round parameters (window lengths, buffers) and the seed rule for `bitIndex`.

11.2 Phase 1: Devnet prototype (end-to-end)

- Implement round lifecycle on Solana devnet: create → commit → finalize → reveal → settle.
- Implement deterministic settlement flows (WIN/LOSE/NO-REVEAL) with full on-chain auditability.
- Provide a reference client that can sign commit/reveal messages and verify outcomes.
- Publish a public verifier script that replays a round from on-chain data and evidence.

11.3 Phase 2: Oracle quorum and evidence hardening

- Stand up at least two independent messenger/oracle operators.
- Enforce quorum finalization with stored attestation metadata.
- Standardize evidence publication format (content-addressed objects + hashes).
- Add monitoring dashboards and public logs for oracle behavior and finalization timelines.

11.4 Phase 3: Scalability optimizations

- Add batching improvements (commit/reveal arrays) and benchmark compute usage.
- Explore compact commitment storage (e.g., aggregated commitments) to reduce per-ticket state.
- Stress-test with large sample sizes and publish performance metrics and cost profiles.

11.5 Phase 4: Security maturation

- Complete at least one independent security review and address findings publicly.
- Introduce bug bounty and responsible disclosure process.
- Harden oracle membership controls and consider bonded/slashing mechanisms if required.

11.6 Phase 5: Mainnet launch (conservative)

- Launch with conservative parameters (windows, quorum) and clear risk disclosures.
- Establish transparent treasury reporting and operational budgets.
- Define an upgrade posture (time-locked upgrades, bounded parameter changes) and publish changelogs.

11.7 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:

- missed reveal windows (NO-REVEAL forfeiture),
- client-side mistakes (incorrect signing, reused salts, wrong nonces),
- network congestion or service interruptions,
- reliance on third-party relayers or front-ends.

12.4 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.

12.5 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.

12.6 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.

12.7 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
token decimals	0
bitIndex mode	per-ticket deterministic
commit window	TBD
reveal window	TBD
oracle quorum	TBD

B Message Schemas

List exact schemas (domain separated) for: Commit, Reveal, Oracle Attestation, and replay protection fields.