Commit Protocol

A Reputation-Aware, Time-Locked Escrow & Verification Standard for the Agentic Economy

Technical Whitepaper

Commit Protocol Team
`Hackathon Submission v0.1`

January 2026

Abstract

The rapid adoption of Decentralized Autonomous Organizations (DAOs) has exposed a critical infrastructure gap: the lack of a trustless, standardized mechanism for verifying and settling complex work. Current solutions rely on manual multisig transactions or subjective bounty boards, leading to payment latency, dispute friction, and scalability bottlenecks.

Commit Protocol introduces a novel primitive: the *Commit.* A Commit is an optimistic, time-locked escrow smart contract that leverages off-chain AI Agents as objective witnesses. By combining automated verification evidence with a reputation-weighted staking model, the protocol aligns economic incentives to make honest work effortlessly settled while making malicious disputes prohibitively expensive.

This paper outlines the system architecture, the "Basic-Judge" agent framework, the dynamic staking mathematics, and the specific implementation details of the prototype developed for this hackathon.

Contents

1 Introduction

The decentralized economy has successfully financialized assets (DeFi) and art (NFTs), but it has failed to efficiently financialize *work*. Despite managing billions in treasury assets, DAOs operate on "handshake agreements" and web2-style trust.

1.1 The "Trustware" Gap

In traditional corporations, legal contracts and employment laws provide the enforcement layer for work. In DAOs, code is law, but "work" (e.g., writing code, designing interfaces, marketing) happens off-chain, making it invisible to the blockchain. This disconnect creates four systemic failures:

1. **Settlement Latency:** Contributors often wait weeks for signers to manually review work and execute transactions.
2. **Subjective Disputes:** Without a predefined acceptance standard, disputes devolve into emotional arguments (e.g., *"I don't like this design"*).
3. **The Griefing Vector:** Malicious actors can spam low-effort submissions (Sybil attacks), or DAOs can refuse to pay for valid work (Rug-pulling).
4. **Fragmented Reputation:** A contributor's reliability history is trapped in centralized silos (Discord, Slack), making it impossible to assess risk programmatically.

1.2 The Commit Solution

Commit Protocol solves this by introducing **Optimistic Agentic Settlement**. We invert the traditional model: instead of requiring human approval to *release* funds, the protocol automatically releases funds after a time lock, *unless* a dispute is raised. To prevent spam disputes, we introduce a **Dynamic Staking Model** where the cost to dispute is determined by the contributor's on-chain reputation and AI-verified confidence scores.

2 Market Analysis & The Agentic Shift

The market for decentralized work is currently bifurcated between "Bounty Boards" and "Gig Marketplaces." Both are ill-equipped for the coming wave of AI-generated contributions.

2.1 Competitive Landscape

Feature	**Upwork/Fiverr**	**Gitcoin/Dework**	**Kleros**	**Commit Protocol**
Settlement	Centralized Escrow	Manual Multisig	Arbitration Only	**Optimistic Smart Contra**
Verification	Human Review	Human Review	Human Jury	**AI Agent Witness**
Dispute Cost	High (Support Tix)	Social Friction	Flat Fee	**Dynamic Staking**
Reputation	Siloed (Database)	Siloed (Profile)	None	**On-Chain Vector**

Table 1: Comparison of Work Settlement Layers

2.2 The Rise of Agentic Work

As of 2025, over 30% of code commits in open-source repositories are AI-assisted. By 2027, it is projected that autonomous agents will perform the majority of DAO operational tasks (ops, data entry, basic coding). Current platforms require humans to verify every task. This is unscalable. Commit Protocol is designed as the **payment rail for the agentic economy**, allowing an AI Agent to submit work and another AI Agent (the Auditor) to verify it, with humans only intervening in the 0.1% of contested cases.

3 System Architecture

The protocol is composed of three distinct layers: the Interface Layer, the Intelligence Layer (Agents), and the Settlement Layer.

3.1 Layer 1: Interfaces

- **Discord Bot (@commitbot):** The primary interface for DAOs. It allows users to Create, Fund, and Submit work directly where they collaborate. It is a stateless client that signs strict intent commands.
- **Web Dashboard:** A Next.js application that serves as the "Explorer" for Commits. It visualizes the Evidence Packages (e.g., visual diffs, test logs) pinned to IPFS.

3.2 Layer 2: The Intelligence Fleet (Agents)

This layer consists of specialized "Auditor Agents." Unlike generic LLMs, these are purpose-built scripts that act as objective witnesses. They do not decide outcomes; they generate **Evidence**.

3.2.1 The "Basic-Judge" Framework

Every agent follows a strict input/output schema called the *Basic-Judge Routine*:

1. **Input:** The ‘spec‘ (from IPFS) and the ‘submission‘ (URL/Hash).
2. **Process:** Fetch data from external APIs (GitHub, Figma, X).
3. **Verify:** Run deterministic checks (CI pass) and probabilistic checks (LLM analysis).
4. **Output:** A signed JSON payload containing a `confidenceScore` (0.0-1.0).

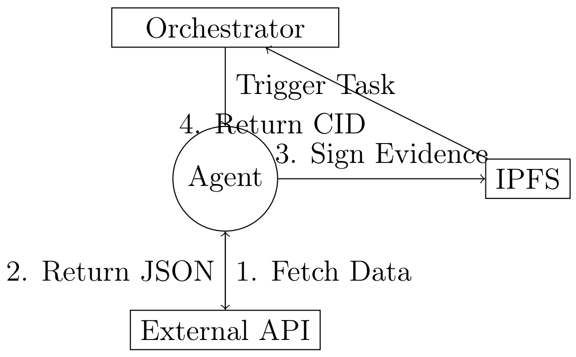

Figure 1: Agent Verification Flow

3.3 Layer 3: Settlement (On-Chain)

The `Commit.sol` contract is the financial core. It enforces the state machine:

$$\text{CREATED} \rightarrow \text{FUNDED} \rightarrow \text{SUBMITTED} \rightarrow (\text{SETTLED} \vee \text{DISPUTED})$$

Crucially, the contract **does not know** about the AI agents. It only knows: 1. Has the deadline passed? 2. Has a dispute been opened with the correct stake?

This separation of concerns ensures that a failure in the AI layer does not freeze funds; it simply reverts the system to a manual dispute process.

3.4 Data Availability Strategy

To maintain low gas costs, Commit Protocol utilizes a "Hash-and-Pin" architecture.

- **On-Chain:** Only the IPFS CID of the Evidence Package is stored.
- **Off-Chain:** The actual logs, screenshots, and diffs are pinned to IPFS (via Pinata or Web3.Storage).
- **Verification:** The Orchestrator cryptographically signs the CID before submission, ensuring the evidence cannot be tampered with after the agent produces it.

4 Economic Security: The Dynamic Stake

Traditional systems use a fixed fee for disputes. This is inefficient: it is too expensive for small tasks (discouraging justice) and too cheap for large tasks (encouraging griefing). Commit Protocol utilizes a **Dynamic Staking Formula** driven by Reputation and AI Confidence.

4.1 Reputation Vectors

Reputation is not a single number. It is stored as a vector $\vec{R}$ for each address:

$$\vec{R} = \langle V_{total}, N_{commits}, S_{rate}, D_{lost} \rangle$$

Where:

- V_{total}: Total value (in USD) successfully settled.
- $N_{commits}$: Number of completed commits.
- S_{rate}: Settlement rate (Commits / Disputes).
- D_{lost}: Number of disputes lost (slashing events).

4.2 The Staking Formula

The Required Stake (S_{req}) to open a dispute is calculated as:

$$S_{req} = S_{base} \times M_{time} \times M_{rep} \times M_{AI} \tag{1}$$

4.2.1 Derivation of Multipliers

1. Time Multiplier (M_{time}): Prevents "Sniper Disputes" where a malicious actor waits until the last second to dispute, forcing the contributor to miss a deadline.

$$M_{time} = 1 + \left(0.5 \times e^{-\lambda t}\right)$$

Where t is the time remaining in days. As $t \to 0$, M_{time} increases.

2. Reputation Multiplier (M_{rep}): Protects high-reputation contributors. If a contributor has a perfect track record, disputing them implies a higher probability of the disputer being wrong (or malicious).

$$M_{rep} = 1 + \frac{\log(V_{total} + 1)}{K}$$

Where K is a scaling constant (e.g., 10,000).

3. AI Confidence Multiplier (M_{AI}): Leverages the Agent fleet. If the AI is 99% sure the work is correct, the dispute cost doubles.

$$M_{AI} = \begin{cases} 2.0 & \text{if } C \geq 0.95 \\ 1.5 & \text{if } 0.80 \leq C < 0.95 \\ 1.0 & \text{if } C < 0.80 \end{cases}$$

4.3 Game Theory Analysis

- **Nash Equilibrium:** The dominant strategy for a Contributor is to submit valid work to ensure M_{AI} is high. The dominant strategy for a DAO is to let valid work settle to avoid paying S_{req} and losing it in arbitration.
- **Sybil Resistance:** A new account with $V_{total} = 0$ has $M_{rep} = 1.0$, making them cheaper to dispute. This discourages creating fresh accounts to spam low-quality work.

4.4 Slashing Mechanics

In the event of a dispute, the losing party forfeits their stake. The distribution of the slashed stake is defined as:

- **50%** to the Winning Party (Compensation).
- **30%** to the Arbitrator (Service Fee).
- **20%** to the Protocol Treasury (MNEE Buyback/Burn).

This ensures that disputing is never "zero-cost," even if you are right—incentivizing off-chain resolution before on-chain escalation.

5 Security Architecture

5.1 Operational Security (Relayer)

The system relies on a "Keeper" or "Relayer" to execute the `settle()` function when a deadline passes.

- **Trustless Execution:** `settle()` is a public function. Anyone can call it. If the official Relayer goes down, the Contributor can manually call the function on Etherscan to release their funds.
- **Key Management:** The official Relayer uses a hot wallet with minimal ETH balances, restricted solely to contract interaction, mitigating the risk of key compromise.

5.2 Attack Vector Analysis

Vector 1: The "Lazy Agent" Attack
Attack: An Agent operator simply signs "Pass" on every submission to save compute costs.
Mitigation: Spot-checks. The protocol randomly selects 1% of submissions for "Deep Review" by a secondary, trusted Oracle. If a discrepancy is found, the Lazy Agent is slashed.

Vector 2: The "Colluding Disputer" Attack
Attack: A Creator and a Disputer collude to lock a Contributor's funds indefinitely.
Mitigation: Time-bound Arbitration. If a dispute is not resolved by the Arbitration Oracle (e.g., Kleros) within X days, the protocol defaults to the outcome with the highest economic weight (Staked Amount).

Vector 3: Oracle Manipulation
Attack: Compromising the Reputation Oracle to artificially inflate M_{rep}.
Mitigation: Reputation updates are batched and signed by a federated set of signers. A single compromised key cannot alter the reputation database.

6 Hackathon Implementation Scope

This whitepaper reflects the architecture of the prototype developed during the hackathon. Below is the breakdown of the current operational capabilities versus future extensions.

6.1 Current Implementation (v0.1)

The following components are fully functional and deployed on testnet:

- **Smart Contract:** `Commit.sol` deployed on Base Sepolia. Handles Create, Fund, Submit, and Dispute flows.
- **Agent Orchestrator:** A Node.js backend that listens for contract events and triggers the GitHub Auditor.
- **GitHub Auditor Agent:** A specialized script that clones the submitted repo, checks for CI success, and uses an OpenAI wrapper to verify basic spec compliance.
- **Discord Interface:** A bot allowing users to create commits via slash commands.

6.2 Future Protocol Extensions

Post-hackathon development will focus on:

- **Decentralized Arbitration:** Replacing the current admin-based dispute resolution with a Kleros Court integration.

- **Visual Agents:** Integrating the Figma API to allow design verification.
- **Reputation Sync:** Indexing historical data from GitPOAP and other on-chain credentials to bootstrap the reputation vector.

7 Conclusion

Commit Protocol is the missing layer in the DAO stack. By formalizing the definition of "Done" through cryptographic commits and agentic verification, we enable a future where DAOs can scale operations without scaling bureaucracy. This hackathon prototype demonstrates the viability of "Optimistic Agentic Settlement" as a new standard for on-chain work.

A Appendix A: Smart Contract Interfaces

```
interface ICommit {
    enum State { CREATED, FUNDED, SUBMITTED, DISPUTED, SETTLED }

    struct Commit {
        address creator;
        address contributor;
        address token;
        uint256 amount;
        uint256 createdAt; // Timestamp
        uint256 deadline;  // Timestamp
        string specCid;    // IPFS Hash
        State state;
    }

    // Core Lifecycle
    function createCommit(
        address _contributor,
        address _token,
        uint256 _amount,
        uint256 _duration,
        string calldata _specCid
    ) external returns (uint256 commitId);

    function submitWork(
        uint256 _commitId,
        string calldata _evidenceCid
    ) external;

    // Dispute Logic
    function openDispute(uint256 _commitId) external payable;

    // Settlement
    function settle(uint256 _commitId) external;
}
```

Listing 1: ICommit.sol Interface

B Appendix B: Evidence Schema Detail

The following JSON schema is enforced by the Orchestrator for all Agent submissions.

```
{
  "$schema": "http://commit.protocol/schemas/evidence-v1.json",
  "meta": {
    "commitId": "0x123...abc",
    "agentId": "agent-github-v1.2",
    "timestamp": 1789400200,
    "signature": "0xabc...123"
  },
  "analysis": {
    "confidenceScore": 0.95,
    "passFail": true,
    "metrics": {
      "testCoverage": 88.5,
      "securityIssues": 0,
      "specMatch": "High"
    }
  },
  "artifacts": [
    {
      "type": "log",
      "url": "ipfs://QmLogHash...",
      "label": "CI Run Output"
    },
    {
      "type": "diff",
      "url": "ipfs://QmDiffHash...",
      "label": "Visual Regression Overlay"
    }
  ]
}
```

Listing 2: Standard Evidence Payload