Mimic Protocol

February 2025

Blockchain Automation Protocol

Facundo Spagnuolo
CTO
facu@mimic.fi

Daniel Fernandez
CEO
daniel@mimic.fi

Abstract

Mimic Protocol is a decentralized automation protocol designed to streamline, standardize, and scale task execution in blockchain. Built on years of iterative development and real-world experimentation, it introduces a modular, multi-layered architecture that ensures reliability, security, and transparency for users and developers alike. The protocol consists of three layers:

- **Planning Layer**: Enables users to define and schedule deterministic tasks using oracle-signed inputs and relayer-verified execution proofs.
- **Execution Layer**: Optimizes execution using intents by coordinating solvers, prioritizing proposals based on efficiency and reliability while enforcing robust reward and penalty mechanisms.
- **Settlement Layer**: Integrates directly with smart contracts to enforce execution conditions and ensure trustless interactions.

Through its innovative use of consensus mechanisms, reputation-based scoring, and economic incentives, Mimic Protocol addresses critical challenges in automation, including decentralization, fault tolerance, scalability, and ease-of-use. This whitepaper consolidates the learnings from Mimic's automation journey into a publicly accessible protocol that empowers the community with efficient and trustless automation.

1 Introduction

Since its inception in 2022, Mimic has been at the forefront of automation in the ecosystem. Through years of iteration, experimentation, and deep engagement with the challenges of trustless environments, Mimic has evolved into a comprehensive automation protocol. This document represents the culmination of our journey—a protocol designed to standardize, scale, and democratize automation, making it accessible to users, developers, and infrastructure providers alike.

Mimic Protocol is built from the ground up to address the complexities and inefficiencies inherent regarding automation. By integrating robust task planning, secure intent execution, and fail-safe mechanisms, it transforms fragmented processes into a seamless and reliable system. Mimic leverages a multi-layered architecture that ensures user-defined tasks are executed with precision, transparency, and scalability, offering a unified solution to the growing demands of a rapidly evolving blockchain landscape.

This whitepaper outlines the key design principles, technical foundations, and innovative features of Mimic Protocol, demonstrating how it empowers users and developers to automate workflows securely and efficiently. Our goal is to not only push the boundaries of automation but also establish a new standard for interoperability, reliability, and accessibility in blockchain.

2 Glossary

2.1 User

The entity that defines and submits tasks to the network. A user specifies the task logic, required input data, and execution trigger. Users are the primary initiators of network activity and are responsible for setting up tasks securely.

2.2 Task

A task is the fundamental unit of execution defined by a user within the protocol. Each task consists of three components:

- **Inputs**: Data required for execution, typically fetched from oracles. Inputs are validated by relayers using cryptographic signatures provided by oracles.
- **Logic**: A deterministic function specified by the user that processes the inputs and evaluates conditions to decide whether an intent should be generated.
- **Trigger**: A configuration defining when the task should be executed (e.g., based on time intervals, specific events, or user-defined conditions).

Tasks act as the bridge between **Relayers**, who execute the task logic, **Oracles**, who provide the necessary inputs, and a central coordinator called **Axia**, in charge of receiving the generated intents if the task conditions are met. By combining these components, tasks enable automation and precise evaluation of user-defined criteria within the protocol.

2.3 Intents

An intent is a structured request that specifies an action to be executed, subject to predefined conditions and constraints cryptographically signed. Intents serve as the core units of work within the protocol, enabling decentralized systems to process, validate, and execute user-defined operations.

2.4 Relayers

Relayers are decentralized network participants responsible for executing tasks on behalf of users. They fetch oracle-signed data, evaluate the task logic deterministically, and submit proofs of execution to the protocol. Relayers ensure that tasks are executed faithfully, adhering to the user-defined conditions and inputs.

Relayers are incentivized through rewards for valid executions and are penalized for invalid or malicious activity. They play a crucial role in maintaining the integrity and reliability of the network.

2.5 Oracles

Oracles are decentralized data providers responsible for supplying accurate, cryptographically signed inputs required for task execution. Oracles respond to queries from relayers with verified data, including values, timestamps, and cryptographic proofs. This ensures that all task executions are based on trusted and reproducible inputs.

Oracles are rewarded for providing valid responses within the expected range and are penalized for outliers or inconsistent data. They serve as the backbone of the protocol's data integrity, enabling deterministic task execution.

2.6 Axia

Axia is the central coordinator within the protocol responsible for managing the life-cycle of intents execution. It validates intents, broadcasts them to eligible solvers, evaluates solver proposals, and ensures robust execution by prioritizing the best proposals. Axia also tracks solver performance and enforces penalties or rewards to maintain reliability and efficiency in the system.

2.7 Solvers

Solvers are entities that compete to fulfill intents. They respond to intent broadcasts with proposals detailing execution parameters, including fees, output amounts, and estimated completion times. Solvers are evaluated based on reputation, execution fees, and timeliness. Reliable solvers are rewarded, while those that fail to execute valid proposals are penalized.

2.8 Settler

The Settler is the protocol component responsible for executing validated intents on-chain, ensuring that user-defined safeguards are respected and finalizing the outcome based on the winning proposal selected through the solvers network. It acts as the final step in the life-cycle of an intent, transforming it from a validated request into a concrete action.

2.9 Safeguards

Safeguards are cryptographically signed constraints defined by the user to impose strict controls over the behavior and execution of an intent. Safeguards ensure that intents are executed within specific, user-defined boundaries, preventing misuse, unauthorized actions, or undesired outcomes.

3 Architecture overview

The Mimic protocol consists of three core layers: the **Planning Layer**, the **Execution Layer**, and the **Security Layer**. Each layer involves key actors and components working together to ensure deterministic task execution, reliable intent handling, and robust security for any kind of user operations. Below is a detailed breakdown of each layer and its responsibilities:

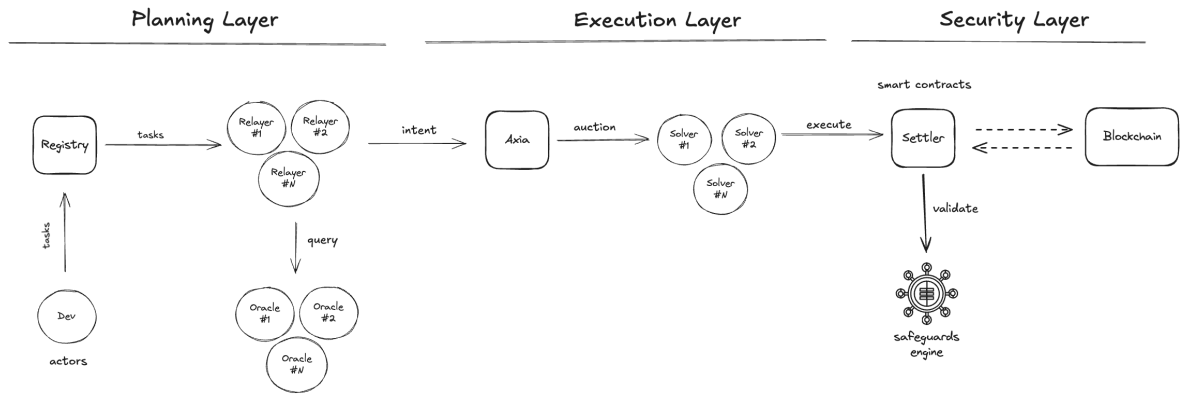

Figure 1: Mimic Protocol Architecture

3.1 Planning Layer

The Planning Layer is the starting point of the protocol. It allows users to define **tasks**—deterministic units of logic that evaluate predefined conditions. A task specifies the inputs it needs, the execution trigger, and the logic that processes those inputs to decide whether an intent should be generated. For instance, a user may create a task to monitor the price of a token and generate an intent to sell if its value exceeds a specific threshold.

Relayers are responsible for executing these tasks. Acting as decentralized operators, they fetch the required inputs from oracles, execute the user-defined logic deterministically, and submit the results back to the system. If the task conditions are met, relayers generate intents, which are then sent to the Execution Layer for further processing.

The oracles play a vital role in ensuring that tasks are based on reliable data. They provide cryptographically signed inputs, such as token prices, account balances, or other blockchain-related data. These inputs are validated to ensure they fall within acceptable ranges or consensus mechanisms, such as medians or averages. By integrating signed oracle data, the Planning Layer ensures that tasks are deterministic and verifiable.

In essence, the Planning Layer connects the efforts of users, relayers, and oracles to generate intents—user-defined requests to execute specific operations in the protocol. These intents form the output of the Planning Layer and serve as the primary input for the Execution Layer.

3.2 Execution Layer

The Execution Layer acts as the central coordinator for processing intents. Once an intent is generated in the Planning Layer, it is handed off to **Axia**, the protocol's central engine. Axia is responsible for validating the intent to ensure it complies with user-defined constraints and broadcasting it to a network of solvers.

Solvers, competitive entities within the system, respond to intents with execution proposals. Each proposal details how the solver intends to fulfill the intent, including fees, execution parameters, and estimated outcomes. Axia evaluates these proposals based on predefined criteria—such as execution fees, solver reputation, and timeliness—and selects the most optimal proposal through an auction-like process. The winning solver then executes the intent on-chain and submits proof of execution back to Axia.

This layer ensures that user-defined operations are executed efficiently while maintaining flexibility and scalability. By enabling competition among solvers, the Execution Layer maximizes efficiency and minimizes costs for users.

3.3 Security Layer

The Security Layer is the backbone of the protocol, ensuring that all operations are executed securely and reliably. At the heart of this layer is the **Settler**, a smart contract that validates the execution proofs submitted by solvers. The Settler ensures that the solver's execution aligns with the original intent and user-defined constraints.

In addition to validation, the Settler facilitates the flow of tokens into and out of the system. It manages user deposits, withdrawals, and other token movements, ensuring that funds are only released when the conditions of the intent are fully satisfied.

Complementing the Settler is the **Safeguards Engine**, a protective mechanism that enforces additional conditions and prevents malicious or erroneous transactions. For example, it can ensure that a transaction does not exceed predefined slippage limits or violate other user-defined parameters.

Through this layer, the protocol provides strong guarantees of security and correctness, safeguarding user funds and maintaining trust in the system.

4 Planning Layer

The Network is the part of Mimic protocol that ensures the reliable execution of user-defined tasks in a decentralized environment. By leveraging relayers to perform deterministic computations, the Network incentivizes correct execution while introducing robust verification and reward mechanisms. The Network handles execution securely through deterministic proofs and oracle-signed inputs, with penalties for misbehavior.

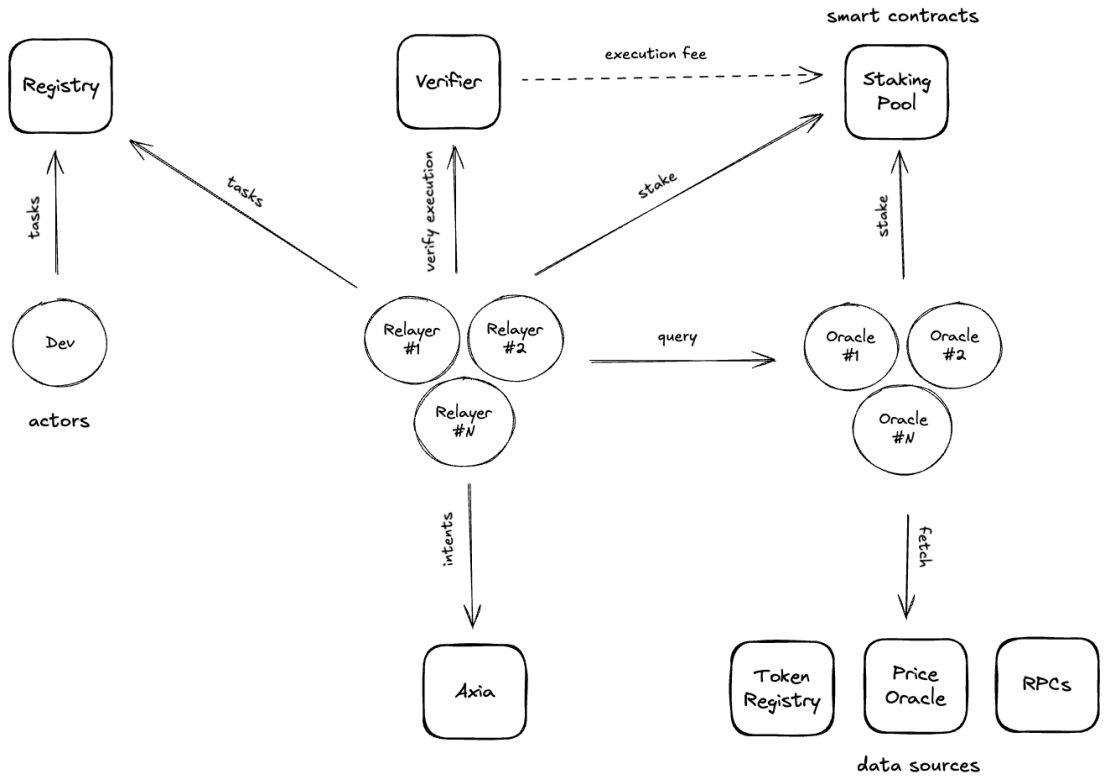

Figure 2: Mimic Protocol Planning Layer

4.1 Overview

The Network involves the following entities:

1. **Users**: Define tasks with input requirements, execution logic, and execution trigger.
2. **Relayers**: Compete to execute tasks correctly and submit proofs for rewards.
3. **Oracles**: Provide signed data as inputs to ensure deterministic task execution.
4. **Verifier**: Validate execution proofs, reward relayers and oracles, and enforce penalties.

Key features of the Network include:

1. **Deterministic Execution**: Tasks depend solely on signed inputs and deterministic functions.
2. **Verification Mechanisms**: Ensures correctness via oracle data and reproducible results.
3. **Economic Incentives**: Rewards for correct execution and penalties for invalid proofs.
4. **Scalability**: Tasks can run in any language, ensuring flexibility for developers.

4.2 Task Definition

A task T is defined by users as a tuple with the following components:

$$T = (I, f, d)$$

Where:

- I is the input schema defining required data, provided by oracles.
- f is the deterministic function for task execution.
- d is the trigger defined for the task.

Tasks are public through an on-chain registry where users can show their support to signal relayers these can be tracked. The input schema is defined as:

$$I = \{i_1, i_2, \ldots, i_n\}$$

Where each input i_k is the result of applying the consensus mechanism c_k to the query function q_k, across the oracle responses from $j = 1$ to m:

$$i_k = c_k(q_k), \quad q_k = \{r_{k,1}, r_{k,2}, \ldots, r_{k,m}\}, \quad r_{k,j} = (\mathrm{v}_{k,j}, \mathrm{t}_{k,j}, \mathrm{sig}_{k,j})$$

Where:

- m is the number of confirmations required for each input.
- c_k is the consensus mechanism applied to q_k.
- q_k is the set of oracle responses for the query function k.
- $\mathrm{v}_{k,j}$ is the j-th oracle response value for the query function k.
- $\mathrm{t}_{k,j}$ is the j-th oracle response timestamp for the query function k.
- $\mathrm{sig}_{k,j}$ is the j-th oracle response signature for the query function k.

4.3 Triggers

The execution trigger defines how often or under what conditions tasks within Mimic Protocol are executed. This mechanism ensures that automation occurs precisely when required, minimizing unnecessary operations and optimizing resource usage. Mimic Protocol supports three primary trigger types: **event-driven triggers**, **cron-based schedules**, and **custom conditions**.

4.3.1 Events

Event-driven triggers leverage blockchain events to execute tasks whenever specific conditions are met. These triggers are particularly useful for automation requiring precise, real-time responses to changes in blockchain state, such as:

- Tasks can be triggered by events emitted from contracts, like token transfers or swaps.
- Detect events on one chain and execute corresponding operations on another.

This approach allows tasks to react dynamically to blockchain activity, reducing unnecessary polling and ensuring timely execution.

4.3.2 Cron Schedules

Cron-based schedules enable tasks to execute at regular intervals, using standard cron syntax. This method is well-suited for recurring operations where real-time responsiveness is less critical, such as:

- Aggregating token prices or other data periodically.
- Performing regular maintenance tasks like rebalancing portfolios or checking system health.

Cron schedules provide a simple yet flexible mechanism for deterministic, time-based task execution.

4.3.3 Custom Conditions

Future iterations of Mimic Protocol aim to support custom state conditions as triggers. This enhancement will allow tasks to execute only when specific blockchain states are met, such as:

- Triggering a task when an account's balance exceeds a defined threshold.
- Activating an operation when a contract's TVL surpasses a certain amount.

While custom conditions provide highly efficient and tailored execution, they pose challenges in tracking and validation. When implemented, they will significantly enhance performance and resource optimization from the user's perspective by reducing unnecessary task executions.

4.4 Oracle Queries

Oracle queries provide the data required for task execution within the protocol. Each query type has a corresponding consensus mechanism to ensure deterministic, accurate, and tamper-proof results. Below are examples of common query types and their associated consensus mechanisms:

4.4.1 RPC Calls

RPC calls provide static and block-related data from the blockchain, such as account balances, contract states, or transaction details. These calls are tied to specific block numbers, ensuring consistency across results. As discrepancies are rare in these queries, the proposed consensus mechanism is the mode, which selects the most frequently occurring value among the responses:

$$c_k(q_k) = \text{mode}(\text{v}_{k,1}, \text{v}_{k,2}, \ldots, \text{v}_{k,m})$$

Since RPC calls are block-specific, any deviation in responses is typically indicative of an error, and the mode ensures the most consistent and accurate result is selected.

4.4.2 Prices

Token prices are fundamental for DeFi operations, such as swaps, liquidations, or rebalancing. Unlike RPC calls, token prices are not inherently tied to a specific block number and may vary slightly due to market conditions. To mitigate the influence of outliers and ensure robustness, the proposed consensus mechanism is the median:

$$c_k(q_k) = \text{median}(\text{v}_{k,1}, \text{v}_{k,2}, \ldots, \text{v}_{k,m})$$

The median effectively isolates extreme values, ensuring that the consensus price reflects the majority of valid responses, even in volatile market conditions.

4.4.3 Input submission

This is the first step of the execution process, the goal is to ensure all relayers use the same inputs before proceeding with the execution per se. Inputs are submitted and validated in this step. Once a relayer detects at a specific time t a task T can be executed based on its trigger d, it can proceed to fetch the oracle-signed responses $r^t_{k,j}$ from $j = 1$ to m. The verifier will essentially validate every oracle response as follows:

$$\text{validate}(r^t_{k,j}) = \begin{cases} \text{True}, & \text{if verifyTrigger}(d, \text{t}^t_{k,j}) \wedge \text{verifySig}(r^t_{k,j}) \\ \text{False}, & \text{otherwise} \end{cases}$$

Where:

- d is the trigger defined for the task.
- $\text{t}^t_{k,j}$ is the oracle response timestamp associated to $r^t_{k,j}$.
- $\text{verifyTrigger}(d, \text{t}^t_{k,j})$ is the function that validates $\text{t}^t_{k,j}$ is valid for the trigger defined d.

Applying the corresponding consensus mechanisms to each type of input, the canonical inputs for task T at time t can be defined as follows:

$$I^t_{\text{C}} = \{i^t_1, i^t_2, \ldots, i^t_n\}$$

4.4.4 Execution submission

Once the input submission succeeds, relayers can execute the task T deterministically using the validated oracle responses to ensure a consistent output. Once the output is computed, relayers can produce their proof of execution as follows:

$$P_r^t = (H_T, I_{\mathrm{C}}^t, O_r^t, C_r^t, \mathrm{sig}_r)$$

Where:

- H_T is the hash of the task T.
- I_{C}^t is the canonical input defined at the input submission time t.
- O_r^t is the output produced by T executed by the relayer r at the input submission time t.
- C_r^t is the complexity in seconds required by the relayer r at the input submission time t.
- sig_r is the signature of the relayer r for the given execution parameters.

Relayers submit their proof of execution hashed to avoid influencing other relayers:

$$\mathrm{hash}(P_r^t, \mathrm{salt}_r)$$

This process remains open until a minimum number of proofs has been submitted.

4.4.5 Execution verification

Once the minimum number of proofs has been reached, relayers must reveal their proofs content. Each proof of execution is validated as follows:

$$\mathrm{validate}(P_r^t) = \begin{cases} \mathrm{True}, & \text{if verifySig}(P_r^t) \\ \mathrm{False}, & \text{otherwise} \end{cases}$$

For a task execution to be considered valid, at least R_{min} relayers must submit identical outputs O_r^t.

Let $\mathcal{R}(O^t)$ be the set of relayers whose outputs match O^t:

$$\mathcal{R}(O^t) = \{r \mid O_r^t = O^t\}$$

The consensus condition is satisfied if:

$$|\mathcal{R}(O^t)| \geq R_{\min}$$

If there is no consensus, the execution is considered invalid and all the relayers are penalized.

4.4.6 Rewarding mechanism

Based on the consensus mechanism $c_k(q_k)$, the set of valid oracles is defined as follows:

$$\mathcal{O}_{\mathrm{valid,k}} = \{j \mid \mathrm{v}_{k,j} \in [c_k(q_k) - \Delta\mathrm{v}, c_k(q_k) + \Delta\mathrm{v}]\}$$

Each oracle $j \in \mathcal{O}_{\mathrm{valid,k}}$ receives the reward $R_k^{\mathcal{O}}$ corresponding to the query q_k.

The set of valid relayers is defined as follows:

$$\mathcal{R}_{\mathrm{valid}} = \{r \mid r \in \mathcal{R}(O^t)\}$$

Each relayer $r \in \mathcal{R}_{\mathrm{valid}}$ receives the following reward:

$$R_r^{\mathcal{R}} = R_b^{\mathcal{R}} \cdot \begin{cases} \mathrm{m}/C_r^t, & \text{if } C_r^t > \mathrm{m} + \Delta\mathrm{c} \\ \mathrm{m}, & \text{otherwise} \end{cases}$$

Additionally, the relayer that has submitted the input signed by the oracles is rewarded by:

$$R^I = \sum_{i=1}^{n} R_b^I \cdot m_i$$

4.4.7 Slashing mechanism

Based on the consensus mechanism $c_k(q_k)$, the set of invalid oracles is defined as:

$$\mathcal{O}_{\text{invalid,k}} = \{j \mid \text{v}_{k,j} \notin [c_k(q_k) - \Delta\text{v}, c_k(q_k) + \Delta\text{v}]\}$$

Each oracle $j \in \mathcal{O}_{\text{invalid,k}}$ is penalized as follows:

$$P^{\mathcal{O}}_{k,j} = P^{\mathcal{O}}_{k,b} + \gamma \cdot |\text{v}_{k,j} - c_k(q_k)|$$

The set of invalid relayers is defined as:

$$\mathcal{R}_{\text{invalid}} = \{r \mid r \notin \mathcal{R}(O^t)\}$$

Each relayer $r \in \mathcal{R}_{\text{invalid}}$ is penalized with $P^{\mathcal{R}}$.

5 Execution Layer

Axia is the part of Mimic protocol designed to optimize and guarantee the execution of user intents. By acting as a central coordinator, Axia receives user-defined intents, broadcasts them to the network of solvers, and ensures robust execution by prioritizing the most efficient proposals. The protocol incorporates fault tolerance, redundant mechanisms, and a reputation-based scoring system to mitigate flaky solvers and ensure user satisfaction.

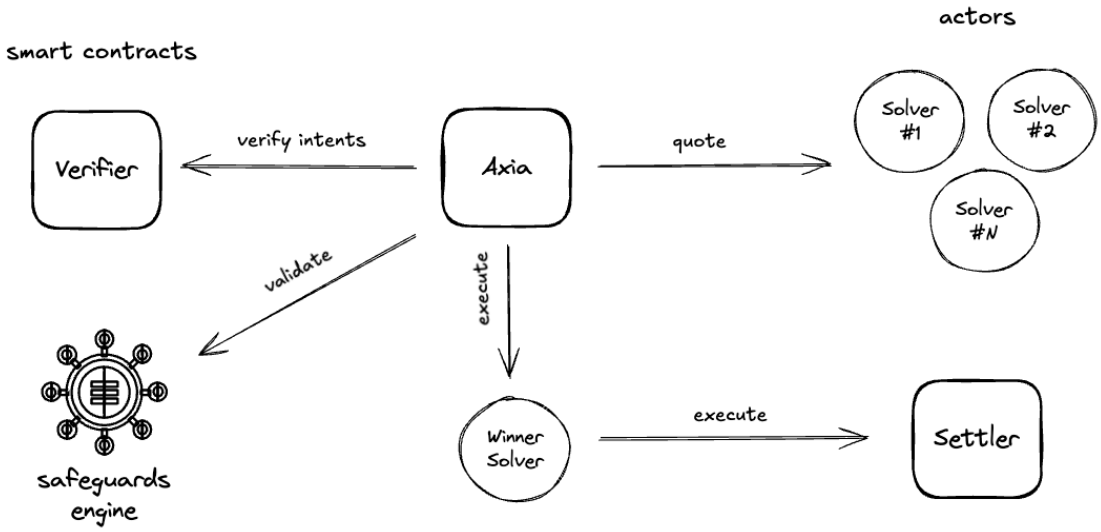

Figure 3: Mimic Protocol Execution Layer

5.1 Overview

Axia operates as a central coordinator that interacts with three main entities:

1. **Users**: Define intents with specific constraints and expectations.
2. **Solvers**: Compete to fulfill these intents by submitting proposals.
3. **Axia**: Ensures the intent is executed by the best possible proposal.

Key features of Axia include:

1. **Guaranteed Execution**: Intent is always executed if valid.
2. **Fault Tolerance**: Handles flaky solvers through retries and fallback mechanisms.

3. **Reputation Scoring**: Tracks solver reliability to prioritize high-performing solvers.
4. **Transparency**: Provides a clear framework for intent handling, selection, and execution.

5.2 Intent Submission

Once outputs from a task are verified on the execution layer, Axia can determine which intents must be auctioned to solvers. Each intent I is a tuple:

$$I = (\mathrm{s}, \mathrm{f}, \mathrm{in}, \mathrm{out}, \mathrm{n}, \mathrm{d})$$

Where:

- $\mathrm{s}, \mathrm{f} \in \mathbb{A}$ are the settler and sender addresses, respectively.
- in is the input the sender is willing to provide to execute the intent.
- out is the minimum acceptable outcome the sender expects in return.
- $\mathrm{n} \in \mathbb{N}$ is a unique identifier to prevent replay attacks.
- $\mathrm{d} \in \mathbb{N}$ is a UNIX timestamp indicating the intent expiration deadline.

Axia validates the intent to discard invalid ones:

$$\text{validate}(I) = \begin{cases} \text{True}, & \text{if } \mathrm{s} \in \mathcal{S} \land \mathrm{d} > t_{\text{now}} \land \mathrm{n} \notin \mathcal{N} \land \text{verifySafeguards(f)} \\ \text{False}, & \text{otherwise} \end{cases}$$

Where:

- $\mathcal{S}$ is the set of valid settlers.
- t_{now} is the current timestamp.
- $\mathcal{N}$ is the set of already-used nonces tracked to prevent replay attacks.
- verifySafeguards(f) validates the sender's safeguards.

5.3 Broadcasting to Solvers

Axia broadcasts the intent to the solver network. The broadcast message includes a unique identifier hash(I) for tracking. Each solver can respond with a proposal P, defined as:

$$P = (\mathrm{t}, \mathrm{d}, \mathrm{s}, \mathrm{out'})$$

Where:

- $\mathrm{t} \in \mathbb{A}$ is the target execution address.
- $\mathrm{d} \in \text{Hex}$ is the solver's execution data in hexadecimal format.
- $\mathrm{s} \in \mathbb{A}$ is the solver's address.
- out' is the output proposed by the solver.

Axia validates proposals using:

$$\text{validate}(P) = \begin{cases} \text{True}, & \text{if out'} \geq \text{out} \land \text{verifySafeguards(f)} \\ \text{False}, & \text{otherwise} \end{cases}$$

5.4 Proposal Selection

Axia evaluates proposals P_k using a composite score S_k:

$$S_k = w_1 \cdot \text{Rep}_k - w_2 \cdot \text{Fee}_k^{\text{norm}}$$

Where:

- $\text{Fee}_k^{\text{norm}}$ is the normalized execution fee.
- Rep_k is the solver's reputation score.
- w_1, w_2 are protocol-defined weight factors.

The normalized execution fee is:

$$\text{Fee}_k^{\text{norm}} = \frac{\text{Fee}_k}{\text{Fee}_{\max}}$$

Where:

$$\text{Fee}_k = \text{fee}(\text{out}, \text{out'})$$

Axia selects the best proposal:

$$P_{\text{winner}} = \text{argmin}_{P_k} S_k$$

Finally, Axia signs the winning proposal:

$$\text{sig} = \text{sign}(\text{hash}(I), \text{t}, \text{out'}, \text{s})$$

5.5 Execution

Axia instructs the winning solver S_{winner} to execute the intent. The solver must:

1. Submit the transaction fulfilling the intent.
2. Return the transaction hash txHash within the execution deadline.

Axia validates execution:

$$\text{validate}(P_{\text{winner}}, \text{txHash}) = \begin{cases} \text{True}, & \text{if txHash fulfills } I \\ \text{False}, & \text{otherwise} \end{cases}$$

If S_{winner} fails, Axia retries with the next-best proposal.

5.6 Rewarding Mechanism

The total reward R is:

$$R = R_b + R_p$$

Where:

- R_b is a base reward for successful execution.
- R_p is a performance-based bonus.

The performance bonus:

$$R_p = w_1 \cdot B_t + w_2 \cdot B_e$$

Where:

- B_t is a bonus for low execution latency.

- B_e is a bonus for efficiency (user receives more than requested).

Latency bonus:

$$B_t = C_t \cdot e^{-x \cdot \frac{\text{latency}_k}{\text{threshold}}}$$

Efficiency bonus:

$$B_e = C_e \cdot e^{-y \cdot \text{Fee}_k}$$

5.7 Slashing Mechanism

Penalties apply for:

1. **Timeouts:**: Solver does not execute in time.
2. **Failures:** Solver fails to execute a winning proposal.
3. **Misbehavior:** Solver submits invalid proofs.

Total penalty P is:

$$P = P_b + P_t + P_f$$

Where:

- P_b is a base penalty.
- P_t is a variable penalty for latency.
- P_f is a variable penalty for execution failures.

Execution failure penalty:

$$P_f = \begin{cases} D_f \cdot n^2, & \text{if failure} \\ 0, & \text{otherwise} \end{cases}$$

Where n is the number of failures in a week.

6 Security Layer

The Settler serves as the final step in the execution of intents within the Mimic protocol. It ensures that intents are executed based on approved solver proposals while enforcing user-defined safeguards. Additionally, it guarantees that the promised outcomes are delivered while protecting against replay attacks and malicious behavior.

6.1 Overview

The Settler validates and enforces the following steps for each intent I and solver proposal P:

1. **Base Intent Validation**: Validates the intent properties, such as nonce uniqueness, deadlines, and signer authenticity.
2. **Proposal Validation**: Ensures the solver's proposal meets the parameters defined in the intent and is signed by an authorized entity.
3. **Safeguard Validation**: Enforces user-defined safeguards embedded within the intent to restrict solver behavior.
4. **Execution**: Executes the solver's proposal and enforces deterministic behavior.
5. **Output Validation**: Confirms that the solver delivered the promised outputs in accordance with the intent.
6. **State Update**: Marks the intent as completed to prevent replay and emits a final execution event.

6.2 Intent Validation

The Settler ensures that the intent I is valid and enforceable by checking:

$$\text{validate}(I) = \begin{cases} \text{True}, & \text{if s} = \text{settler} \land \text{d} > t_{\text{now}} \land \text{n} \notin \mathcal{N} \land \text{verifySafeguards(f)} \\ \text{False}, & \text{otherwise} \end{cases}$$

Where:

- settler is the contract executing the intent I.
- d is the intent deadline.
- t_{now} is the current timestamp.
- n is the intent nonce.
- $\mathcal{N}$ is the set of already-used nonces tracked to prevent replay attacks.
- verifySafeguards(f) validates the sender's safeguards.

If validation succeeds, the nonce n is marked as used:

$$\mathcal{N} = \mathcal{N} \cup \{\text{n}\}$$

6.3 Proposal Validation

The solver's proposal P is validated to ensure it adheres to the user's intent and satisfies all constraints. The Settler checks:

$$\text{validate}(P) = \begin{cases} \text{True}, & \text{if s} \in \mathcal{S} \land \text{out'} \geq \text{out} \land t_{\text{now}} < \text{t} \\ \text{False}, & \text{otherwise} \end{cases}$$

Where:

- $\mathcal{S}$ is the set of authorized solvers.
- out' is the output promised by solver s.
- t is the proposal deadline signed by Axia.

6.4 Execution

If the intent I and proposal P are validated successfully, the Settler proceeds with execution:

1. **Transfer Inputs**: Grants access to the intent's input in.
2. **Execute Proposal**: Calls the solver's execution logic:

$$\text{execute}(P) = \text{call}(\text{t}, \text{d})$$

Where:

- $\text{t} \in \mathbb{A}$ is the target execution address defined in P.
- d $\in$ Hex is the solver's execution data in hexadecimal format.

Finally, the Settler ensures the promised outcome is fulfilled:

$$\text{out"} \geq \text{out'}$$

which ultimately results in transferring the outputs out" to the sender.

7 Economic Incentives

The Mimic Protocol's economic incentives are designed to align the interests of all participants—users, oracles, relayers, solvers, and the community—ensuring trustless and efficient operations. Incentives and penalties are tailored to encourage good behavior while deterring malicious actions, creating a robust and self-sustaining ecosystem.

7.1 Users

Users are the primary initiators of activity within the protocol. They register tasks by staking funds, which serve as a prepaid pool to pay for task execution by oracles and relayers.

7.1.1 Registration

Users must stake a minimum amount of funds to register their tasks. These funds will cover payments to oracles and relayers.

7.1.2 Prioritization

By maintaining a sufficient amount staked, users ensure their tasks are prioritized and executed promptly by relayers and oracles.

7.1.3 Execution

Payments to oracles and relayers are drawn from the user's staked funds when their tasks are executed. Each payment to oracles and relayers contributes to increasing their staking positions, incentivizing consistent and accurate service. Users are responsible for covering protocol fees during task execution, ensuring the ongoing development and sustainability of the network.

7.2 Relayers

Relayers are tasked with executing user-defined tasks and submitting proofs of execution. They are incentivized to act honestly and efficiently, with penalties for malicious behavior.

7.2.1 Staking

Relayers must stake a minimum amount to participate in the network. The stake serves as collateral to cover penalties in case of malicious or incorrect behavior. Relayers wishing to withdraw their stake must undergo a cool-down period during which their activities are reviewed to ensure no unresolved issues exist.

7.2.2 Rewards

Relayers earn rewards for valid task executions. Rewards are drawn from the user's staked funds and increase the relayer's own staking position.

7.2.3 Penalties

If a relayer acts maliciously or submits invalid proofs, their stake is reduced. Relayers found to be consistently unreliable or dishonest may face increased penalties or disqualification.

7.3 Oracles

Oracles provide cryptographically signed data critical for deterministic task execution. Like relayers, oracles are incentivized for honest participation and penalized for malicious behavior.

7.3.1 Staking

Oracles must stake a minimum amount to participate in the network. Their stake acts as collateral to ensure accountability for the data they provide. Oracles must undergo a cool-down period before unstaking their funds, ensuring accountability for their actions.

7.3.2 Rewards

Oracles earn rewards when their responses are validated as part of the consensus mechanism for task execution. These rewards are drawn from the user's staked funds and contribute to increasing the oracle's staking position.

7.3.3 Penalties

Oracles providing invalid or malicious data face penalties that reduce their stake. Repeated violations may result in disqualification from the network.

7.4 Solvers

Solvers are competitive entities in the Execution Layer responsible for fulfilling intents. Their staking mechanism is managed privately within Axia, with penalties and rewards tied to their performance and behavior.

7.4.1 Staking

Solvers maintain a private stake that acts as collateral for their participation. The staking mechanism ensures solvers have a vested interest in fulfilling intents accurately and promptly.

7.4.2 Rewards

Solvers earn rewards for successful intent execution tied to their performance based on criteria such as low latency and high efficiency.

7.4.3 Penalties

Solvers are penalized for misbehavior, including failing to execute winning proposals or submitting invalid transactions. Penalties are deducted from their private stake, and repeated violations can lead to disqualification.

7.5 Community

The Mimic community plays a vital governance role in managing the protocol's evolving needs. Their responsibilities include setting economic parameters, managing intent and query types, and ensuring the long-term sustainability of the protocol.

7.5.1 Governance

Community members propose and vote on protocol settings, including staking thresholds, reward distributions, penalty mechanisms, and intent and query type approvals.

7.5.2 Economic Incentives

Community members may be rewarded for active participation in governance, such as proposing improvements or contributing to the protocol's development. Also, Protocol fees collected from task executions and penalties are partially allocated to community-managed reserves, which can be used for ecosystem growth or maintenance.

8 Protocol Resilience

8.1 Private Execution

Mimic Protocol allows **private execution** to address scenarios where users require a controlled and efficient execution process. This mode allows users to bypass decentralized consensus mechanisms and directly trust a particular relayer or their own relayer mechanism, skipping signature validation and oracle dependencies. It serves as both an operational convenience for specific use cases and a fallback mechanism during catastrophic events.

8.2 Private Settlers

Mimic Protocol allows **private settlers** as an alternative to the shared settlement mechanism, providing users with the ability to isolate their funds from others. Private settlers ensure enhanced security and privacy by keeping user funds separate from the pooled funds of other users.

8.3 Relayers and Oracles Misbehavior Mitigation

Mimic Protocol employs robust confirmation mechanisms and economic incentives to ensure the integrity and reliability of both relayers and oracles. These mechanisms deter misbehavior by aligning economic interests with honest participation.

8.4 Solvers Misbehavior

Mimic Protocol is designed to ensure that solver misbehavior cannot directly impact user funds. The settlement contract acts as a safeguard, enforcing user-defined constraints and validating the execution outcomes before releasing any assets.

If a solver fails to submit a valid transaction, the system retries with the next-best proposal, ensuring intent fulfillment despite occasional solver failures.

8.5 Settler Vulnerabilities

The settler contract serves as the backbone of the Mimic Protocol's security, ensuring that user funds are protected and all execution conditions are strictly enforced.

To address potential vulnerabilities in the settler contract, the protocol implements a governance-controlled switch mechanism that allows execution to be paused immediately in the event of a detected bug or exploit. This ensures a rapid response to mitigate risks without compromising user funds or network integrity.

Acknowledgments

The Mimic Protocol whitepaper has greatly benefited from the contributions and insightful revisions provided by our engineering team. We extend our deepest gratitude to Bruno Balzani, Ramiro Gonzalez, Lautaro Galende, and Pedro Araoz for their technical expertise, feedback, and commitment to refining the protocol's design.

Additionally, we would like to acknowledge the invaluable guidance of our external advisors, Ariel Barmat and Lex Sokolin, whose deep knowledge and thoughtful discussions have significantly shaped the direction of this work.

Their collective contributions have played a fundamental role in strengthening and shaping Mimic Protocol.

A Swap Intents

This appendix provides a detailed explanation of how the protocol supports the specific case of swap intents, using the generalized intent-based protocol defined above. Swaps are a specialized use case where assets are exchanged across different blockchain networks.

A swap intent I can be represented in the same tuple format as the general intent but with specific semantics applied to its fields:

$$I = (\mathrm{s}, \mathrm{f}, \mathrm{in}, \mathrm{out}, \mathrm{n}, \mathrm{d}, \mathrm{sig})$$

Where:

- $\mathrm{in} = \{(c_i, t_i, v_i)\}$ is the list of input token t_i and amount v_i to be provided on the source chain c_i.
- $\mathrm{out} = \{(c_j, t_j, v_j)\}$ is the list of output token t_j and minimum amount v_j to be provided on the destination chain c_j.

Then, each swap intent proposal can be defined as:

$$P = (\mathrm{t}, \mathrm{d}, \mathrm{s}, \mathrm{out'})$$

Where:

$$\mathrm{out'} = \{(t_j, v'_j, c_j) \mid v'_j \geq v_j\}$$

is the list of output token t_j and amount v'_j the solver guarantees to deliver on the destination chain c_j. This allows defining a fee function f as follows:

$$\mathrm{Fee}_k = \frac{\sum_{v=1}^{n} v'_j \cdot \mathrm{slippage}_j}{\sum_{j=1}^{n} v'_j}$$

Where:

$$\mathrm{slippage}_j = \frac{v'_j - v_j}{v'_j}$$

Finally, an atomic settler can be implemented to validate the outcome provided by the solver as follows:

$$\forall v'_j \in \mathrm{out'}, v'_j \geq v_j$$

B Cross-chain Swap Intents

This appendix describes how swap intents can be enforced for cross-chain. One key principle of the Mimic protocol is to ensure atomic intents. This means users will never face an undesired intermediate state. Either their intents are fulfilled or not. This appendix provides a comprehensive description of how cross-chain intents can be fulfilled within Mimic Protocol, ensuring atomicity and security throughout the process.

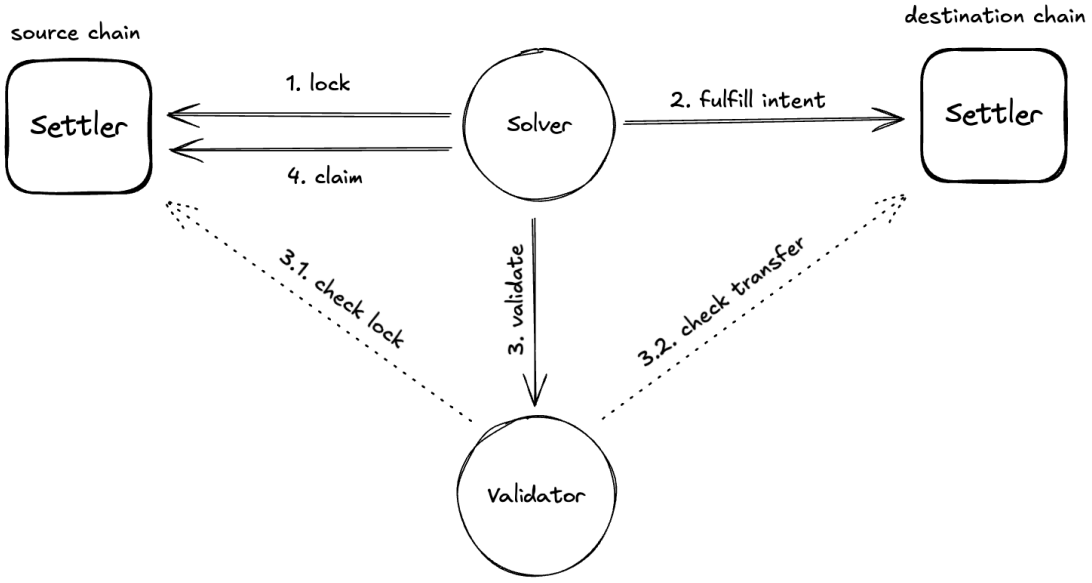

Figure 4: Mimic Protocol Cross-chain Intents

B.1 Terminology and Definitions

B.1.1 Cross-chain Intent

A cross-chain intent is an intent I that holds:

$$\forall c_k, c_l \in \text{in}, \forall c_p, c_q \in \text{out}, c_k = c_l \land c_p = c_q \land c_k \neq c_p$$

B.1.2 Validators

Validators are entities that provide proofs for solvers after verifying they fulfilled the intent on the destination chain. These proofs allow solvers to demonstrate they have accomplished their duty on the source chain.

Note: Mimic Protocol will rely on a Proof of Authority (PoA) committee to serve as validators. Their known identities help ensure trust in the process's integrity.

B.2 Workflow

When processing a cross-chain intent, the following steps must be executed by the solver S_{winner}:

1. Submit the transaction to execute the intent on the source chain.
2. Submit the transaction to fulfill the intent on the destination chain.
3. Require a validator to provide proof that the intent was fulfilled.
4. Submit the transaction to claim the tokens locked on the source chain.

B.2.1 Intent Execution on the Source Chain

On the source chain, the settler locks the intent's input in for a predefined duration T.

$$\text{locked}(I) = \begin{cases} \text{True}, & \text{if } t_{\text{now}} < t + T \\ \text{False}, & \text{otherwise} \end{cases}$$

Where:

- t_{now} is the current timestamp.
- t is the timestamp when the assets were locked.
- T is the duration of the lock in seconds.

B.2.2 Intent Execution on the Destination Chain

During proposal execution on the destination chain, the settler executes P and ensures the promised outcome is fulfilled:

$$\text{out}' \geq \text{out}$$

B.2.3 Intent Fulfillment Validation

The solver provides the corresponding destination chain transaction hash txHash to a validator and requires a proof that the intent was fulfilled.

The validator verifies:

$$\text{validate}(\text{txHash}) = \begin{cases} \text{True}, & \text{locked}(\text{hash}(I)) \wedge \text{txHash fulfills } I \\ \text{False}, & \text{otherwise} \end{cases}$$

Upon successful verification, the validator provides the solver with:

$$\text{sig} = \text{sign}(\text{hash}(I), \text{c}_i)$$

Where:

- $\text{hash}(I)$ is the intent hash.
- c_i is the chain of the intent's input in.

B.3 Claiming Locked Assets

The solver claims the locked tokens by providing a valid proof, which is verified as follows:

$$\text{validate}(I) = \begin{cases} \text{True}, & \text{if verifySig}(\text{sig}, I, c) \\ \text{False}, & \text{otherwise} \end{cases}$$

Where:

- verifySig is the function that validates the signer is an authorized validator.
- c is the current chain.

C Decentralizing Axia

C.1 Introduction

Axia currently operates as a centralized coordinator for executing intents within Mimic Protocol. To further enhance decentralization, security, and fault tolerance, we propose an alternative model in which the coordination duties are distributed among a pool of Axia Coordinators.

C.2 Active Coordinator Selection

A pool of Axia Coordinators is defined as follows:

$$\mathcal{AC} = \{AC_1, AC_2, \ldots, AC_n\}$$

Then, the round-robin selection function is:

$$f : \mathbb{N} \to \mathcal{AC}, \quad f(r) = AC_{((r \bmod n)+1)}$$

This function assigns the coordinator for round r. The initial active coordinator for intent I is then:

$$AC(I) = f(r_0)$$

C.3 Incentives

To maintain system integrity, each coordinator's performance is logged. Any failure to act within the time window or any malicious behavior is recorded. Coordinators who fail to produce a valid confirmation or who are flagged for misconduct will be penalized through reductions in their staked collateral.