Aergo Oracle solution

Gum Protocol

Introduction

For the sustainable growth of a real DeFi ecosystem, managing information that exists outside the blockchain itself is essential. On the AERGO network, the lack of reliable oracle solutions currently limits the potential for large protocols to thrive.

Additionally, centralizing off-chain information sources introduces a critical attack vector, as any system built on a centralized solution would rely on a single point of failure. Past flashloan attacks and DEX exploits involving rapid buy and sell operations highlight the need for a truly decentralized oracle network.

To address this, **Gum Protocol** was created. Its governance token, GUMM, will serve to guide the project through a decentralized decision-making process and serves as a reward for node operators who keep both a high reputation and high-quality data services.

Technical implementation

Gum Protocol consists of four main components: three smart contracts for managing requests, aggregation, and staking, along with node management software. The implementation details are outlined below:

Smart Contracts

a. **Request Contract**: This contract allows users or developers to request information from the oracle network. The request can involve several types of data, ranging from asset prices to real-world data such as weather or hardware information. The request contract facilitates communication between the user and the nodes.

b. **Aggregator Contract**: Once a request is made, an event is triggered on the blockchain to notify the nodes. These nodes respond with the requested data. The aggregator contract then processes these responses, choosing a result based on the nature of the data—either selecting a single response or calculating an average (in cases such as asset prices)—and returns it to the user.
c. **Main Contract**: This contract handles managing the staking process for each node, distributing rewards to active nodes, and penalizing malicious behavior. It tracks the staked GUMM tokens, adjusts node reputation scores, and enforces penalties such as slashing stakes and temporary bans for nodes that provide manipulated or false data.

Node Management Software

The node management software, built in Golang, will handle the data retrieval process, including managing API URLs and authentication keys when necessary. The use of Golang will enable a lightweight and cross-platform executable, allowing for the deployment of multiple instances from a single server, thus facilitating the operation of several nodes concurrently.

Due to its simplicity, an HTML interface could be developed, assuming there is sufficient demand. Alternatively, a .cfg file could serve as a practical means of configuration.

The node management software runs as follows:

First, each node keeps a private key, which will be used as the owner of the contract. At configurable intervals or on-demand, the node can fetch external information and send it to the contract, updating its values.

To achieve this, the node will listen for events from the request contract. Each time a user requests information, the node will update its data accordingly. For this purpose, the node will use the API via gRPC.

Trust Management with Proof of Stake

To manage trust within the network, the GUMM token will be used in a proof-of-stake mechanism, governed by a smart contract that assigns a reputation score to each node. This main smart contract will handle staking, rewards distribution, and penalties for providing incorrect or manipulated data.

Node operators must stake a variable amount of GUMM tokens. If a node is reported for providing false or manipulated information, users can submit a report to the main contract. The contract owner will then review the report and decide whether the node was indeed responsible for data manipulation. If found guilty, the operator's stake will be slashed as a penalty, and the node will be temporarily banned from the oracle network for 24 hours. The contract will also automatically reduce the node's reputation score, making it more difficult for the operator to regain trust.

After the ban period, the operator will have the opportunity to re-stake their tokens and resume providing data.

Each node will have a dynamic reputation score reflecting its performance and reliability. This score enables users to request data directly from specific nodes if they prefer, providing an added layer of flexibility. In such cases, the end-user or developer will pay a direct fee to the chosen node operator, compensating for the individualized service.