DeSoda--Focus on releasing Web3 Social Media interoperation potential

By DeSoda Team on Q2 2024

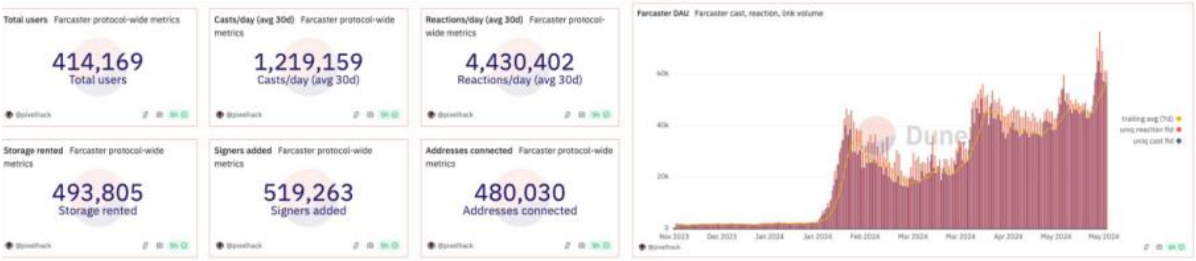
Abstract

In this paper, we introduced DeSoda, a decentralized execution layer, named Soda Layer, that aims to unlock the interactivity of farcaster social graph data, and an optimistic coprocessor solution for Web3 Social Media named Soda coprocessor. First, Soda Layer provides Web3 application developers with a classic Blockchain + VM execution layer to build DApps, and can easily access farcaster social graph data in a VM environment. Before Soda Layer, these data were stored by off-chain Hubs. At the same time, Soda Layer uses the most advanced Eigen Staking AVS as the system consensus module, making full use of EigenLayer's powerful compatibility with Intersubjective digital tasks, giving the protocol unparalleled security. Secondly, through Soda coprocessor, Dapps in any chain can obtain historical social graph data in any Web3 Social Media at a low cost through this decentralized solution, thus bringing more "DeSocial +" application scenarios.

1. Introduction

1.1. Background

With the continuous development of technology, more and more Web3 Social Media has been widely adopted. Among them, Farcaster has achieved rapid growth in a short period of time with its excellent product design. According to Dune Analytics, as of May 28, 2024, the total number of Farcaster users has reached 414,169, and the 7-day average DAU exceeds 60k... These data are exciting and prove that users recognize their products. But at the same time, we also see some room for optimization.

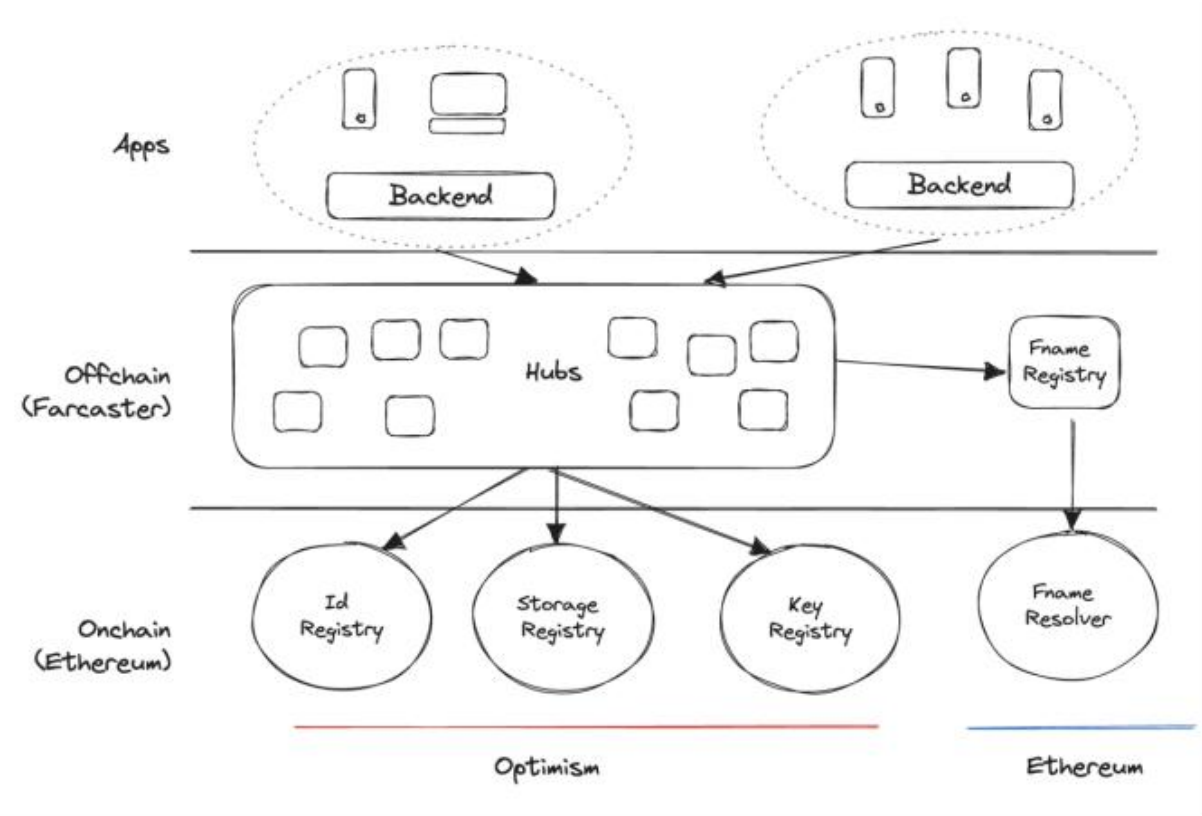


We know that the technical architecture of farcaster is as follows, which consists of two parts, on-chain smart contracts and off-chain Hubs. The on-chain smart contracts mainly deal with three aspects;

- **Id Registry**：The IdRegistry lets users register, transfer and recover Farcaster accounts. An account is identified by a unique number (the fid) which is assigned to an Ethereum address on registration. An Ethereum address may only own one account at a time, though it may transfer it freely to other accounts. It may also specify a recovery address which can transfer the account at any time.

- **Storage Registry**：The Storage Registry lets accounts rent storage by making a payment in Ethereum. The storage prices are set by admins in USD and converted to ETH using a Chainlink oracle. The price increases or decreases based on supply and demand.

- **Key Registry**：The Key Registry lets accounts issue keys to apps, so that they can publish messages on their behalf. Keys can be added or removed at any time. To add a key, an account must submit the public key of an EdDSA key pair along with a requestor signature. The requestor can be the account itself or an app that wants to operate on its behalf.



Hubs is a distributed server network that is mainly responsible for storing and verifying farcaster-related data, and through protocols, ensures that these data meet the CRDT feature (conflict-free replicated data type), that is, each node in a distributed system maintains a copy, and can update the copy independently and in parallel without causing conflicts. In farcaster Hubs, six sets of CRDT data are maintained between nodes, including:

- **Cast CRDT**: This dataset manages the posts of users in farcaster;

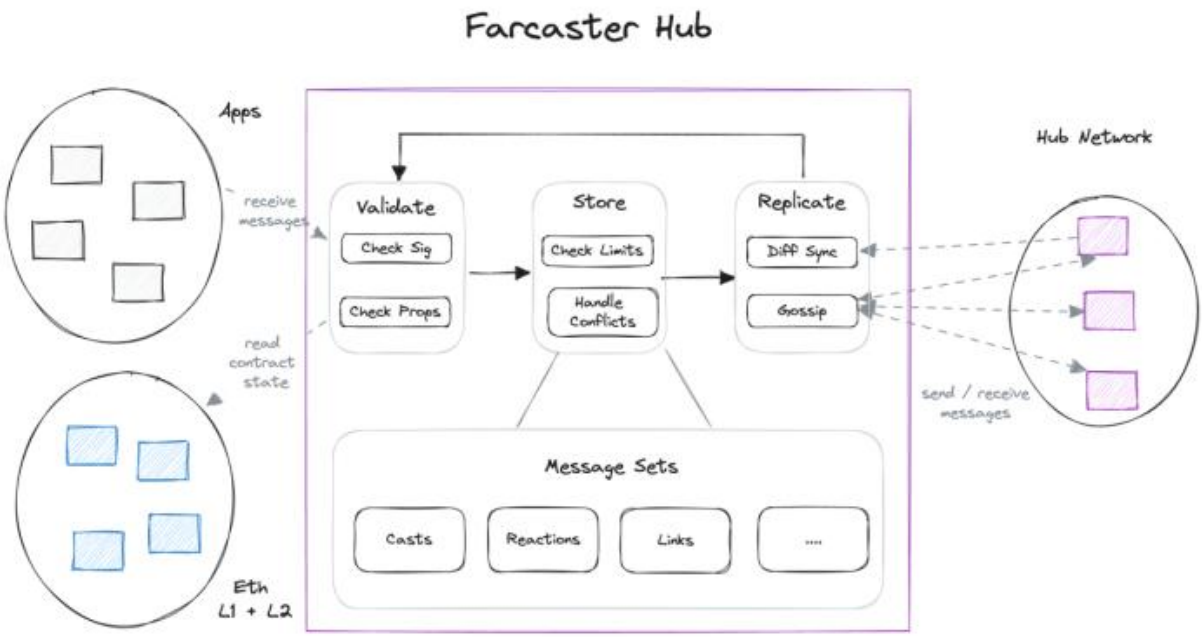
- **User data CRDT**: This dataset manages the profile data of users, including pfp, bio, username, etc.;

- **Reaction CRDT**: This dataset manages the relationship between users, such as like, recast.

- **Verification CRDT**: This dataset manages the cryptographic proof of the user's ownership of a certain Ethereum standard address

- **Links CRDT**: This dataset manages the follow relationship between users, such as A follow B;

- **Username Proof CRDT**: This dataset manages the ECDSA signature of the user's ownership of fName or ENS.



It can be seen that in the current architecture design of farcaster, the user's social graph data is maintained by Hubs. The so-called social graph data refers to the social relationship information of users in social media, such as the followers of user A, which casts user B likes, etc., while the chain only manages key information such as account permissions. Although it is understandable that this is a compromise design for performance and product experience, it also brings some troubles to DApps developers, because Hubs does not provide developers with a trustless execution environment that can obtain this part of information at a low cost.

This greatly hinders the application innovation potential around farcaster. Take Degen, a project with great influence in the current farcaster ecosystem, as an example. This is a tips system in farcaster. Users can reward the publisher of a cast with Degen tokens as a tip by entering the reward amount in the comment of a cast. In order to achieve this function, it has to choose a centralized solution, that is, to carry the monitoring of user reward behavior and the calculation of the amount of Degen airdrops obtained through a centralized server. Because the key information required for these services is maintained by Hubs, this brings centralized risks to the application.

Therefore, we realized a pain point that has not been met in the farcaster ecosystem. Developers designed a trusted execution environment that can easily access farcaster social graph data, or allow the existing trusted execution environment to easily obtain this data. This is the original intention of DeSoda's design.

1.2. Core Idea

As time came to April 29, 2024, Eigenlayer released its Eigen token white paper, which introduced a new AVS type called EIGEN Staking, which gave us inspiration. We know that EigenLayer cleverly expanded Ethereum's consensus through the design of restaking, bringing a cheap and secure consensus layer. Web3 applications can introduce Ethereum's economic security to their applications by developing AVS, bringing the benefits of trustlessness to the protocol and improving their own credibility. However, since the restaking type of AVS needs to reuse the ETH assets staked in Ethereum PoS, there are high requirements for the design of the fraud proof model of AVS, which is only applicable to so-called objectively attributable digital tasks. This limitation was broken with the introduction of EIGEN Staking. Eigenlayer provides a trusted consensus layer protocol for Intersubjective digital tasks by designing an on-

chain forkable work token called Eigen and the corresponding EIGEN Staking model. We believe that this brings a low-cost and efficient solution to the above pain points.

Therefore, based on EIGEN Staking, we designed a trustless trusted execution environment called Soda Layer that can easily access farcaster social graph data. This execution environment is compatible with EVM, and developers can access the latest and final farcaster user social graph data in smart contracts, as well as an optimistic coprocessor solution for Web3 Social Media, called Soda coprocessor. Web3 developers can asynchronously access farcaster's social graph data in a trustless manner in any chain.

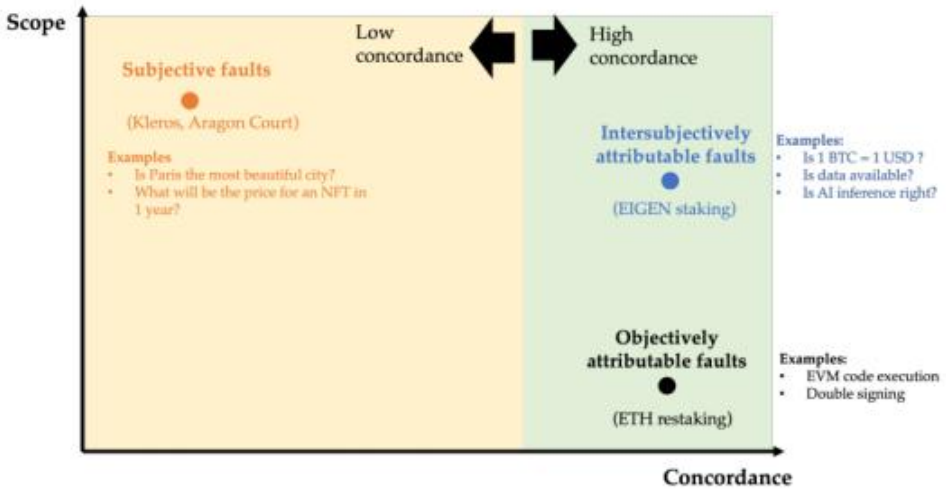
2. DeSoda--Focus on releasing Web3 Social Media interoperation potential

2.1. Sequence the hubs messages is the intersubjective digital task

For Soda Layer, the most intuitive understanding is to sort the farcaster hubs messages through the consensus service provided by Eigen Staking, so that it has a chain structure, and through the system function, the VM can access this data. So the first thing to discuss is whether EIGEN Staking is suitable for this scenario. According to Eigenlayer, EIGEN Staking is suitable for processing intersubjective digital tasks. The so-called intersubjective digital task refers to a digital task in which there is a broad and consistent judgment standard for whether the execution result is correct in the group. According to the different types of errors that may occur, it is divided into two types:

- Errors that can be identified at any time by looking back at past data, such as in the price oracle, the spot price of BTC in Binance is $1 at 00:00:00 UTC on May 8, 2024. This error can be identified at any time afterwards.

- Errors that can only be observed in real time, such as malicious censorship, assuming that a transaction is maliciously refused to execute by a certain node group for a long time.



This brings us design inspiration. We believe that bringing finality to all CRDT data sets of users in Hubs at a certain moment is an intersubjective digital task, because all honest nodes have a consistent judgment on how to judge the correctness of the message that triggers the CRDT update.

Taking Cast CRDT as an example, Hubs triggers the update of the data set through CastAdd and CastRemove messages. There are clear and consistent regulations on how to judge the correctness of the received message: m.signer must be a valid signer message.fid in the signer CRDT additional set. If there is a CastAdd message and a CastRemove message, and their m.hash and m.data.body.target_hash are the same, or there are two CastRemove messages, and their m.data.body.target_hash and are the same, a conflict will occur. The conflict is resolved by the following rules:

- 1. If m.data.type is different, discard the CastAdd message.

- 2. If m.data.type is the same and m.data.timestamp values are different, discard the message with the lower timestamp.

- 3. If m.data.timestamp and m.data.type values are the same, discard the message with the lower lexicographic order.

So it is feasible to bring finality to CRDT datasets in Hubs through EIGEN Staking.

2.2. Soda Layer——A decentralized trusted execution layer based on EIGEN Staking and compatible with farcaster social graph data

Next, let's introduce how Soda Layer does this. Soda Layer AVS consists of six core modules:

- **Hub Relayer Model:** This module is responsible for running a Farcaster Hub node and continuously listening to the message information in Hubs.

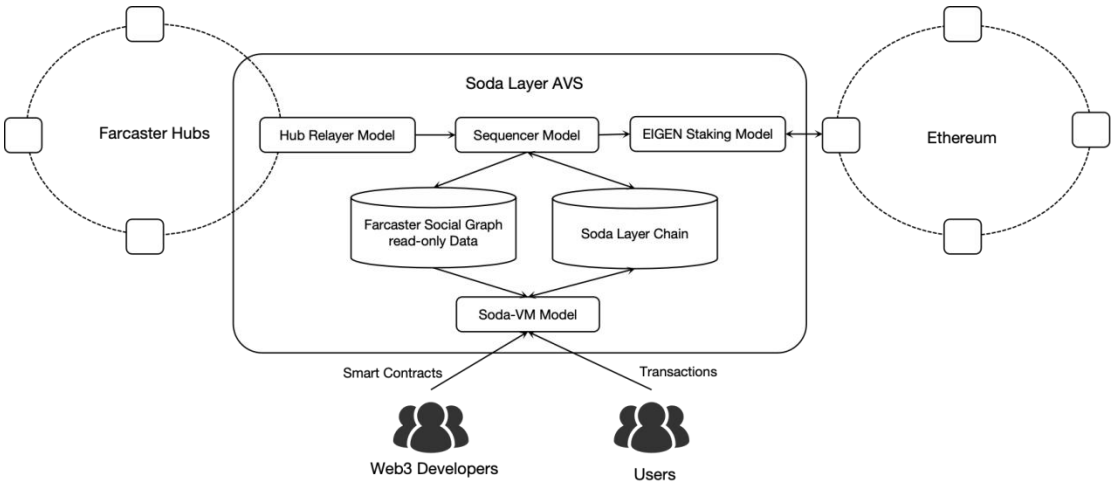
- **Sequencer Model:** This module is responsible for processing two parts. One is to maintain a Social Graph Patricia Tree based on the message information listened to in the Hub Relayer Model. The root node of this data structure is the current social graph status data corresponding to each farcaster user. The second is to sort the transactions of the Soda Layer and package these two parts into a Soda Block.

- **EIGEN Staking Model:** This module is responsible for finalizing the confirmed Soda Block according to the rules of EIGEN Staking.

- **Social Graph read-only data:** This module is responsible for storing and managing the latest farcaster user social graph status data. This part of data is read-only for Soda Layer users and is only updated according to the message in the farcaster Hubs.

- **Soda Layer Chain:** This module is responsible for storing and maintaining Soda block data.

- **Soda-VM Model:** This module is the execution environment of the soda layer. The module is designed as a pluggable architecture and is compatible with mainstream standards such as EVM and WASM.



2.3. Subjective and observed timestamps of messages

We know that Hubs uses messages as the update source of CRDT data set, and it is optimistic about the timing of messages, that is, it recognizes the accuracy of the timestamps actively submitted by users. Although Hubs has a 15-minute tolerance for the timestamp delay attached to the received message, that is, when the Hub receives a message that is more than 15 minutes old, it will be discarded, but this usually cannot meet the requirements of decentralized systems for the degree of confidence. Therefore, in Soda Layer, it does not rely on the subjective timestamp provided by the user in the message for sorting, but relies on the active observation results of the messages propagated in the Farcaster network by the sorter node at regular intervals as the basis for sorting. This is consistent with most blockchain systems. All transactions in the same block have the same timestamp, which is the time when the block packager completes the block packaging rather than the time when the transaction is submitted to Mempool. We call the time when a message is confirmed by Soda Layer the observed timestamp.

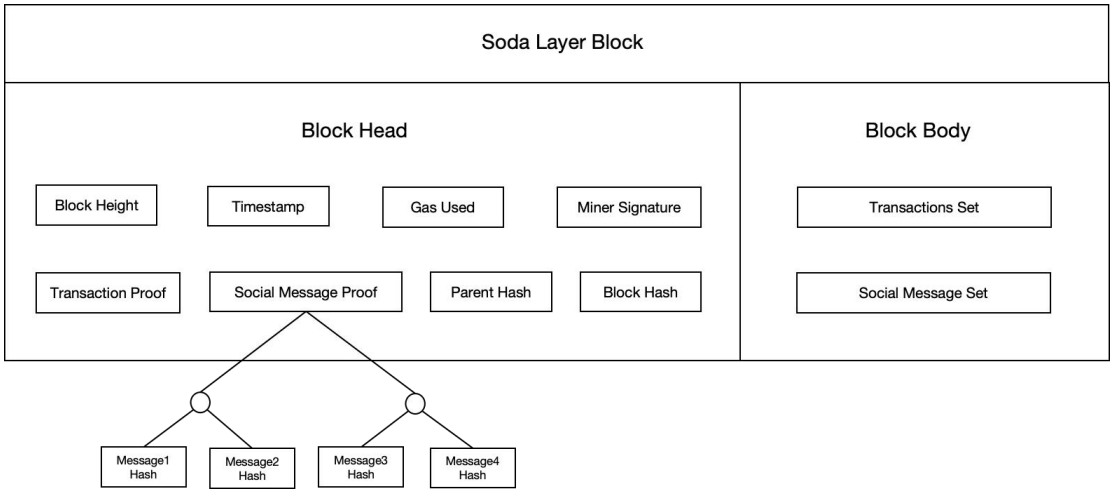
This may have an impact, that is, there may be a delay between the observed timestamp and the subjective timestamp, which will make the logical timing of social data in Soda Layer and social data displayed in Hubs have a certain difference. For example, a user submits a LinkAdd Message at T1 (June 11, 2024 00:00:00 UTC+0), but the timestamp encapsulated in the Message is T2 (June 10, 2024 23:50:00 UTC+0). This will be received in Hubs and widely considered to be a follow of a certain user at T2. However, in the Soda Layer, the follow relationship will only be confirmed after the sorted party actively observes it, and this time will inevitably be later than T1. This is something that DApp developers need to pay attention to.

2.4. Soda Layer Block

The core of the Soda Layer AVS node is to package the Soda Layer Block, where Block Height consists of the following data:

- **Block Height:** current block height;
- **Timestamp:** timestamp of generating this block, which is also the observed timestamp mentioned above;
- **Gas Used:** total Gas usage of transactions packaged in this block;
- **Miner Signature:** signature of the block hash by the generator of this block using his own private key;

- **Transaction Proof:** Merkle Trie Root of all transaction hashes packaged in this block;
- **Social Message Proof:** Merkle Trie Root of all Social Message hashes packaged in this block;
- **Parent Hash:** parent block hash;
- **Block Hash:** total hash of the above information;



The Block Body consists of two parts: a collection of Transactions and a collection of Social Messages. Transactions come from all users of the Soda Layer, while Social Messages are only generated by the sorting nodes that package the blocks based on subjective observations of Hubs.

2.5. Soda Virtual Machine

Soda Virtual Machine is an EVM-compatible execution environment. In the Soda Layer, there is a set of special system smart contracts for accessing Farcaster social network data in Soda Virtual Machines. This provides developers with a trusted execution environment that can access Farcaster social graph data. The pseudo code of the system contract is as follows, and its functions will be continuously expanded as the community needs it. Any user can call these methods to obtain the latest and verified Farcaster social graph information. Before executing Transactions, Soda Virtual Machine will first update the status data in the system contract according to the Social Message Set to ensure the latest social graph information.

```
interface SodaSocialSystemContract {

    // Links related functions

    function getFollowersAmountByFid(address fid) external view returns(uint256 totalAmount);

    function getFollowingAmountByFid(address fid) external view returns(uint256 totalAmount);

    function checkFollower(address targetFid, address beCheckedFid) external view returns(bool isFollower);

    function checkFollowing(address targetFid, address beCheckedFid) external view returns(bool
isFollowing);

    ……
```

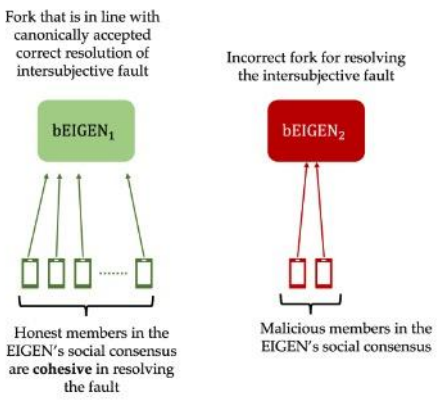
```
    // Reactions related functions

    function getLikesByCastHash(bytes calldata targetCastHash) external view returns(uint256 totalLikes);

    function getRecastsByCastHash(bytes calldata targetCastHash) external view returns(uint256 totalLikes);

    function checkLikes(bytes calldata targetCastHash, address beCheckedFid) external view returns(bool isLiked);

    function checkRecasts(bytes calldata targetCastHash, address beCheckedFid) external view returns(bool isRecasted);

    ……

}
```

2.6. Soda Layer Safety

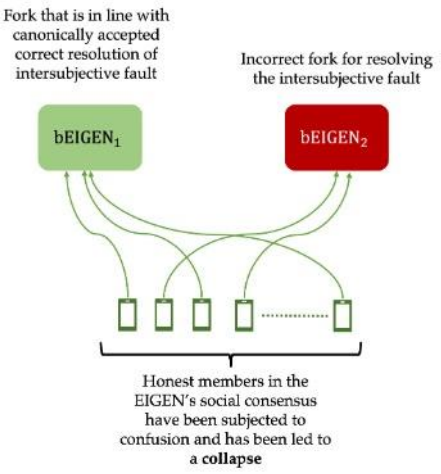
Next, let's discuss the security of the Soda Layer. We know that Eigen Staking requires the design of a comprehensive verification and slashing model for the Intersubjective digital task to help AVS have Cryptoeconomic Safety. The impact of the corresponding Intersubjective attributed fault on AVS can be divided into the following two types:

- **Intersubjective cohesion:** This type of attributable error has the characteristics of intersubjective cohesion, which means that a wide range of AVS nodes or light nodes have the same judgment criteria for the error, so that the correct nodes will have a common standard for the selection of the resulting fork, that is, the right people choose the right fork.

- **Intersubjective fracture:** This type of attributable error carries the risk of intersubjective fracture, which means that a wide range of AVS nodes or request nodes have vague judgment criteria for this type of error, which may lead to disagreements and cause a network fork. This error is common in some marginal conditions, such as due to network delays, some AVS nodes observe BTC prices 5% higher than other nodes.



(a) Intersubjective cohesion.



(b) Intersubjective fracture.

Therefore, it is crucial for Intersubjective Staking AVS to fully consider the possible attributable errors and design slashing mechanisms according to their types, so that they have cryptoeconomic security while taking into account network efficiency and avoiding frequent forks. In the Soda Layer, intersubjective attributed faults can be mainly divided into the following two types:

- Malicious tampering: This is also the most direct malicious behavior, which is to forge a transaction or message, thereby maliciously tampering with the State data. For this problem, since both the transaction and the message have a digital signature of the sender using his private key, this type of problem can be easily identified by verifying the legitimacy of the digital signature. For the miner who packages the malicious block, its staked EIGEN token will be completely confiscated and redistributed to all affected honest users.

- Denial attack: This type of problem is usually that the miner selectively refuses to package certain transactions or messages. For this type of problem, Soda Layer sets up a linear slash mechanism. When a message is found to be delayed for more than 20 minutes and is not packaged by the AVS node, a slash can be initiated against the miner. The slash ratio will be proportional to the delay time until it is completely confiscated.

2.7. Incentives and Economic Model

We know that operating a Farcaster Hub does not generate any income, which may reduce its decentralization because there is not enough economic incentive for potential maintainers. At the same time, in order to ensure the economic security of AVS, we have designed a token economic model with dynamic release and flywheel effect. Simply put, DeSoda will issue a work token - SODA. EIGEN Staking Operators who honestly maintain AVS will be able to receive SODA rewards in proportion. For details, please refer to the subsequent public economic white paper.

2.8. Soda coprocessor - an optimistic coprocessor that breaks down Farcaster Hubs information silos

An important use case of EigenLayer ReStaking is the optimistic coprocessor. The so-called optimistic protocol processing is an optimistic solution of ZK coprocessor based on Eigenlayer Restaking. It seems that where the concept of ZK coprocessor comes from is not very important. As the name suggests, the original intention of this type of product is to use the zero-knowledge proof algorithm to provide coprocessor services for the current mainstream blockchain system, so that it can offload complex and expensive computing operations to the off-chain execution, and the correctness of the execution results is guaranteed by zero-knowledge proof. The most classic example of this modular idea is the relationship between CPU and GPU. By handing over parallel computing operations such as image processing AI training, which the CPU architecture is not good at, to another independent module GPU for processing, the execution efficiency is improved.

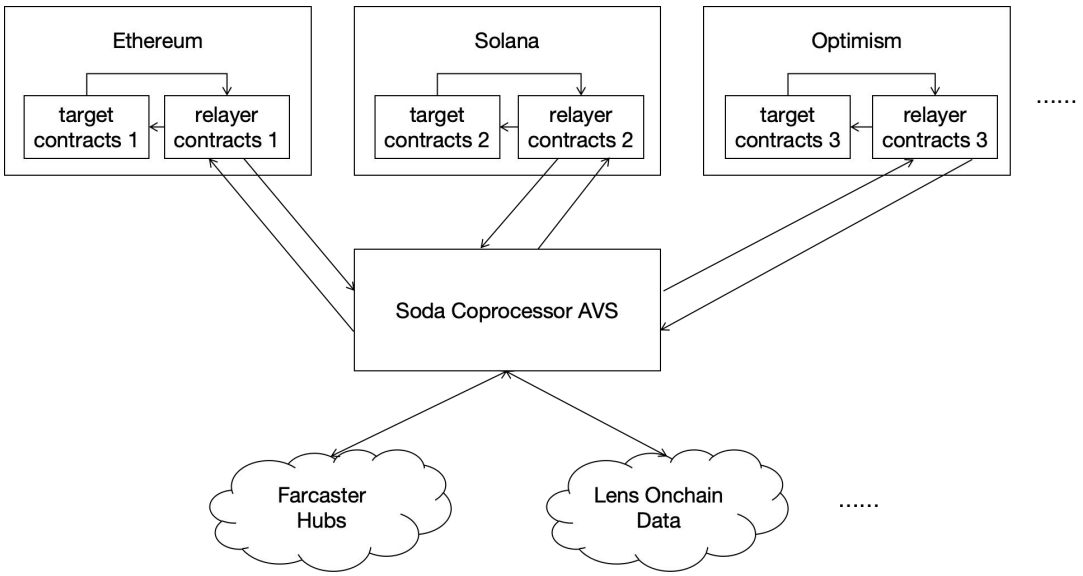
The technical architecture of a classic ZK coprocessor project is basically as follows. Simply put, when a user has a demand for a complex calculation, you can use the off-chain service to calculate the result and generate the relevant ZK Proof, and then use the result and the proof as parameters to call the on-chain verification contract. The contract relies on the execution result, execution proof, and the key block information of the entire chain provided to the chain, such as transaction merkle root (the process of maintaining the key information of the entire chain is also trustless). These three parts of data verify the correctness of the result through the on-chain verification algorithm. After passing the verification, the result will be notified to the target contract through the callback function to trigger subsequent operations.

This asynchronous execution technology solution has not been widely used for a long time due to problems such as high cost, but the emergence of EigenLayer has changed this situation. By outsourcing the verification of ZK Proof to AVS and optimistically trusting the execution results submitted by the AVS node on the chain, the cost of using the coprocessor can be greatly reduced.

2.9. Full-chain Social Data Relay

Since most Web3 Social products have different technical architectures, how to integrate these scattered data at low cost is a crucial issue for most existing DApps. Imagine that Compound wants to design a new credit loan function for its Ethereum product, which hopes to obtain the social media data of all the user's Web3 Social accounts and generate relevant reputation scores. For example, the reputation score is generated based on the weighted sum of the number of fans of the user on Farcaster and the number of fans on Lens Protocol, and the LTV of the user is appropriately increased based on the score. Before Soda Coprocessor, this requirement will not be able to be completed elegantly, because these data are usually distributed in different blockchain systems, or even other distributed systems. Take Lens as an example, its social network data exists in Polygon, while Farcaster exists in Hubs.

With Soda coprocessor, any DApp on the chain can retrieve the social network data of users on the Web3 Social platform in an asynchronous manner. In this process, Soda coprocessor will become the relay of Web3 social data. Back to the example just now, when Compound's new credit loan contract needs to call the total number of fans on Farcaster and Lens of a certain user, it can send a request to the Soda coprocessor Relayer contract with subsequent operations. After receiving the request, Soda coprocessor AVS will query the corresponding data and return it to the corresponding chain through the Soda coprocessor Relayer contract, and trigger subsequent operations through the callback function. In this way, the corresponding social network data can be obtained on the chain through asynchronous calls.



2.10. Use Cases

For developers who want to develop Web3 Social DApps, Soda Layer has a variety of usage scenarios, including:

- **SocialFi**: We know that the most anticipated scenario in the Web3 Social field has always been the SocialFi scenario. Before the Soda Layer, since farcaster only managed some key identity information on the chain, the user's social graph information was not on the chain, there were relatively large

restrictions on the SocialFi scenario based on farcaster. Through Soda Layer, DApp developers can easily use social graph information to unlock more SocialFi scenarios.

- **Marketing:** An important value of social media is marketing. For a long time, the marketing of most Web3 products has focused on x. With the development of farcaster, this situation will gradually shift to farcaster. Therefore, it is considered interesting to design some attractive marketing activities around farcaster. In the past, many project parties' marketing activities were criticized for centralization. Through Soda Layer, designing marketing activities through smart contracts will greatly enhance credibility. For example, product A has designed a fan growth activity. If you follow the official account within a given period of time, you will have the opportunity to participate in a 100 USDT lottery. Such a simple marketing activity can be fully realized through Soda Layer, thereby ensuring the transparency and credibility of the entire process and increasing user interest in

- **Credit DeFi:** There is a major sub-segment in the DeFi scenario that has always been in the spotlight, that is, credit-derived DeFi, such as credit loans. We know that credit in contemporary society is composed of many aspects. In addition to financial situation, your social relationship is also a very important evaluation factor. For example, a KOL with 1 million fans has a relatively higher cost of doing evil, so he has stronger credit. In Soda Layer, since social graph information can be easily accessed, it will be more helpful to build credit-derived DeFi products.

- **DAO**: As a social media platform, a large number of social relationships will be generated in Farcaster, which may lead to more and more DAOs. Then there will naturally be a wide range of management tools around DAOs, such as DAO salary management, reputation system management, etc. These will be easily implemented on the Soda Layer, and can be connected through social graph information to unlock more functions and enrich more scenarios.

Reference

[1] Eigen Labs Team: *EIGEN: The Universal Intersubjective Work Token Towards the Open Verifiable Digital Commons*

[2] Farcaster Docs: https://docs.farcaster.xyz/

[3] Farcaster Specifications:

https://github.com/farcasterxyz/protocol/blob/main/docs/SPECIFICATION.md

[4] Lens Doc: https://www.lens.xyz/docs