Verisense Whitepaper

Verisense Team

Version 2: 2024-08-18

Abstract

This white paper introduces Verisense, a pioneering Validation-as-a-Service (VaaS) network designed to address the critical challenges of pooled security in the burgeoning restaking ecosystem. With fierce TVL wars on the supply side and little attention paid to AVS on the demand side, leading restaking protocols struggle to meet the diverse security and performance requirements of various AVS applications and systems. Verisense offers a scalable and flexible lightweight consensus mechanism that decouples the consensus from runtime at the node level, alongside a developer-friendly SDK, advanced node management, and cutting-edge features like FHE-enabled blind voting and an adaptive slashing mechanism. By positioning itself as an independent module that can be integrated with any restaking platform, Verisense not only enhances security and performance for AVSes but also unlocks new market opportunities for restaking partners. This paper details the mechanisms underlying Verisense's unique approach, including the Monadring protocol and blind voting design, and explores how these innovations empower AVSes to achieve optimal consensus and security while minimizing operational overhead.

Background

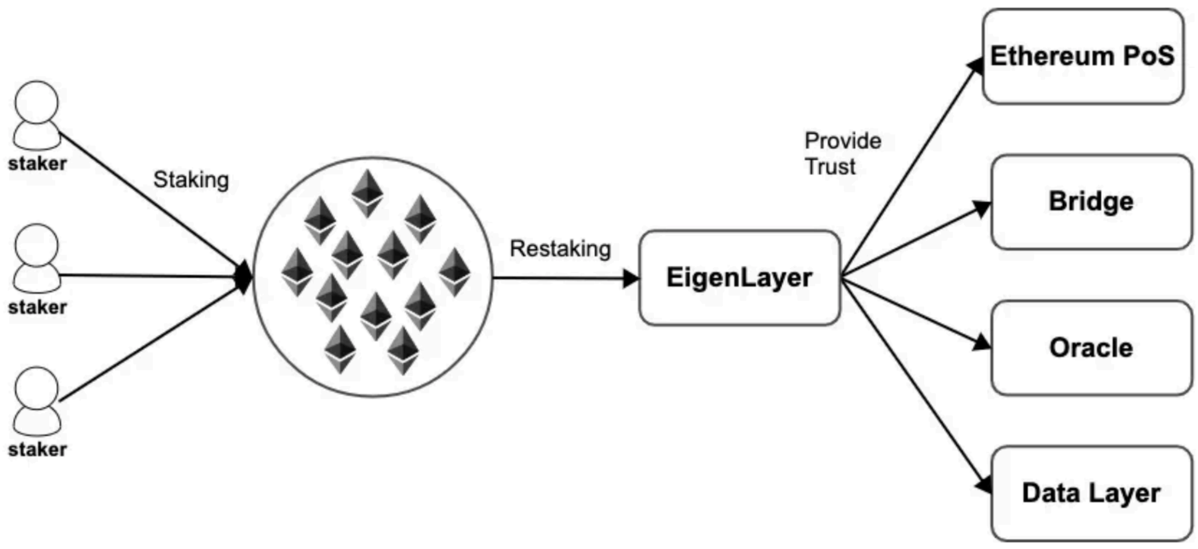
Decentralized applications (dApps) often face challenges in customizing their level of decentralization to suit the importance of their underlying data. For example, decentralized social media applications may require faster data i/o response and lower storage costs compared to decentralized finance (DeFi) applications which typically demand a higher level of decentralization. The ability to adjust decentralization based on application needs is crucial, yet existing blockchain architectures have not fully addressed this flexibility.

However, a lower degree of decentralization introduces higher risks of fraud. In such a setup, each node carries a relatively high weight within the network, making it easier for a few colluding nodes to engage in malicious behavior. In a typical Byzantine Fault Tolerance (BFT)[13] Proof-of-Stake (PoS)[14] network, $\frac{2}{3}+1$ of the nodes must agree on a proposal for it to pass. This requirement becomes problematic when the network is less decentralized, as it increases the likelihood of a few colluding nodes compromising the network's integrity or causing it to halt.

This inherent tension between the level of decentralization and network security leads us to the concept of pooled security. By implementing pooled security, we can meet the demand for customizable levels of decentralization without compromising security.

Restaking with Pooled Security for AVS

EigenLayer, the first protocol to introduce the concept of restaking for pooled security, enhances Ethereum's staking infrastructure by allowing users to "restake" their staked ETH to secure additional networks and applications. This concept has since expanded beyond Ethereum by many following restaking players to include a wide range of valuable token assets as pooled collaterals, both ERC-20 and non-ECR-20. By pooling security across multiple networks, the restaking mechanism raises the cost of potential attacks, thereby strengthening the overall ecosystem. Validators can support various AVSes, aligning economic incentives with security needs, and fostering competitiveness among different protocols.



EigenLayer architecture

By restaking, AVS can leverage pooled security primarily from Ethereum and potentially from other participating networks with additional valuable collateral, all on-demand. This approach involves two critical aspects:

1. **Pooled collateral**: AVS benefits from a restaking mechanism backed by valuable underlying assets as pooled collateral.
2. **On-Demand Security**: Security can be provisioned when needed, providing flexibility.

This dual feature— pooled security from a valuable network and the ability to obtain it on-demand—is what drives the growing interest in restaking. Restaking introduces an economic and social engineering approach, offering greater flexibility and potential for future innovations.

The Problem of the Current Restaking War

Sreeram Kannan, CEO of EigenLayer, describes EigenLayer as a "Verifiable Cloud for Crypto" allowing ETH holders to restake their staked ETH and extend Ethereum’s security to additional protocols and applications, known as AVSes in this context. This approach enables AVSes to leverage Ethereum's secure trust network without needing to establish their own validator sets, effectively commoditizing decentralized trust. EigenLayer functions as a marketplace where developers can incentivize validators to secure their protocols, much like SaaS platforms provide scalable and flexible software solutions.

However, AVSes encompass a broad range of applications and functionalities, each with unique security requirements and performance expectations from the underlying blockchain. EigenLayer's reliance on structuring AVSes as independent nodes within a network—typically a blockchain—introduces several challenges, which are also prevalent in other leading restaking protocols like Karak and Symbiotics:

1. **Flexible and Scalable Consensus:** AVSes have diverse security needs and varying levels of affordability. Meeting these needs requires a consensus mechanism that is both highly flexible and scalable, capable of accommodating different degrees of decentralization and varying network sizes.
2. **Balancing Security and Performance:** While rapid consensus is essential for quick performance, the security of smaller networks is more easily compromised. Necessary measures, such as rotating nodes within the active cluster, are needed to prevent collusion and enhance security.
3. **Fair and Resilient Slashing Mechanisms:** Maintaining network integrity requires a fair and effective slashing mechanism. This mechanism must be capable of monitoring, reporting, penalizing, and publicizing malicious behavior among nodes.
4. **Diversified Requirements:** Some AVSes, particularly those aiming to replace centralized web2 platforms (e.g., dYouTube, dTikTok, dFacebook), demand higher performance and enhanced functionalities such as faster data I/O, more efficient file storage and retrieval, and more frequent message synchronization. General-purpose L1 blockchains currently fail to adequately serve such applications, and the same challenge persists in the restaking context.

Many leading restaking platforms, including EigenLayer, recognize the critical importance of supporting AVSes but struggle to address these challenges:

1. **Limited AVS Variety:** AVSes can be broadly categorized into i) chain-natured (e.g., sequencers, sidechains, oracles), ii) non-chain-natured (e.g., keeper networks, trusted execution environments, threshold cryptography schemes, new virtual machines, decentralized social apps), and iii) hybrid models. EigenLayer primarily focuses on the low-hanging fruit of chain-natured AVSes, neglecting other types of AVSes with more complex requirements.
2. **Rigid Solutions:** The one-size-fits-all approach fails to adequately serve the diverse needs of AVSes, resulting in suboptimal solutions and a lack of paying clients. The true paying demand lies with non-chain-based AVS clients, who lack existing ready-to-use AVS-based infrastructure solutions. This sector is where leading restaking infrastructures struggle most.

3. **Dysfunctional Slashing Mechanism:** The absence of a robust and adaptable slashing mechanism hampers the flow of value between the supply side and the demand side, hindering the efficient operation of the restaking system.
4. **Sustainability Issues:** Relying on token incentives to attract Total Value Locked (TVL) and maintain operators is not a sustainable model. As TVL and the node pool grow, so do the burdens on the network, accelerating the potential failure of this business model.

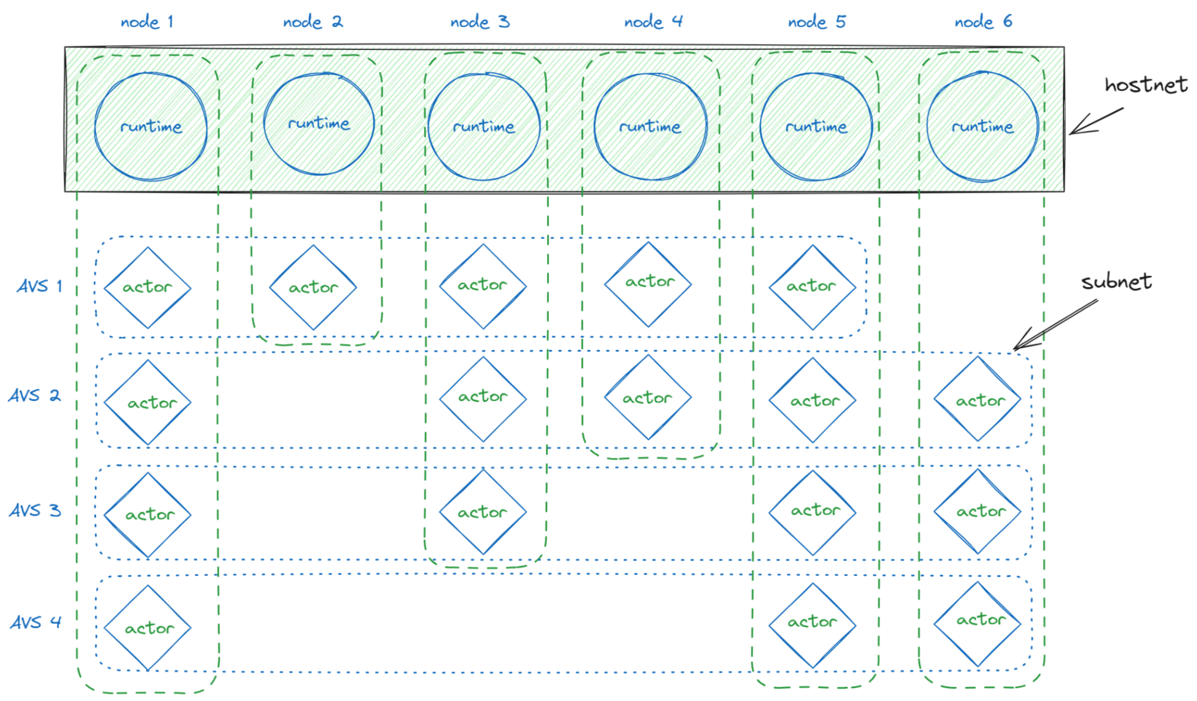
To address these issues, Verisense proposes a lightweight Validation-as-a-Service (VaaS) model that reduces the burden of bootstrapping for AVSes while enhancing security, particularly for networks with a lower degree of decentralization.

Verisense VaaS

In the ongoing restaking war, attention has largely centered on the supply side of pooled security, with major players fiercely competing for Total Value Locked (TVL). However, it is equally important to address the demand side. Verisense shifts this focus by introducing Validation-as-a-Service (VaaS), a hierarchical structure designed to meet the diverse needs of various AVSes with scalability and flexibility. Verisense VaaS operates as an independent module that can integrate with any existing restaking platform, effectively unlocking and catering to AVS demand.

Hostnet and Subnet

Verisense's architecture features two key concepts: **hostnet** and **subnet**, as illustrated in the following figure:



Hostnet and subnet, node and AVS

This structure includes several essential components:

- **Runtime:** The runtime represents the on-chain logic and execution environment for a Verisense node, effectively implementing the blockchain within Verisense.
- **Actor:** An actor is an off-chain task that operates alongside the on-chain executor. It executes specific WebAssembly (WASM) code fetched from the runtime.
- **Node:** A Verisense node (represented as a column) is a fundamental unit of the network, encompassing one runtime and several actors.
- **Hostnet:** The hostnet comprises all runtimes across nodes, forming a singular, unified network.
- **AVS:** An AVS (represented as a row) consists of actors across nodes, with each AVS including one actor from each participating node. Different AVSes can have distinct sets of actors. To enhance security, the actor set within an AVS rotates each epoch.
- **Subnet:** Each AVS functions as a subnet, defined in relation to the hostnet.

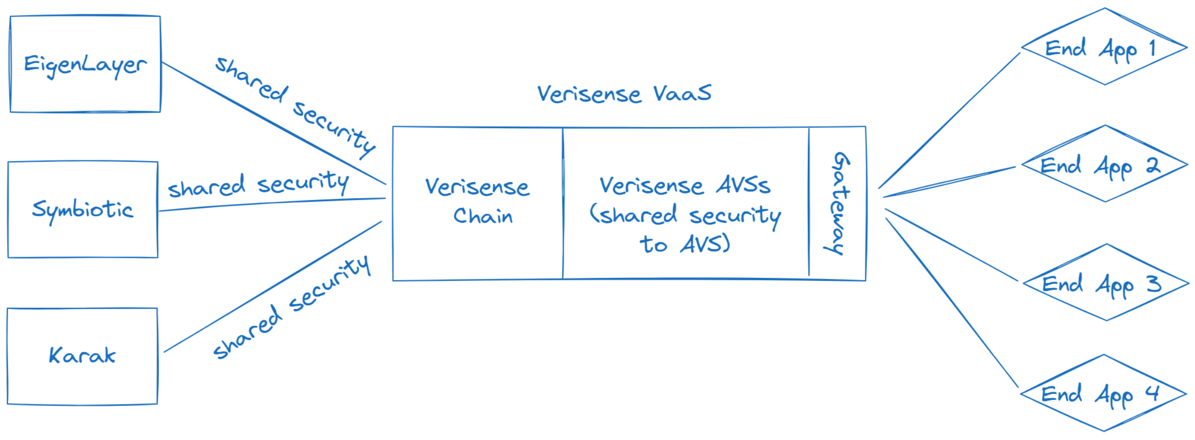
This structure enables Verisense to support hundreds or even thousands of AVSes within its hostnet matrix, providing both flexibility and scalability for a wide range of applications.

A Concise Approach of Pooled Security in the Context of Restaking

To understand the security model of Verisense, let's first consider its architecture. From an external perspective, AVSes operate within the Verisense network, benefiting from its overall security.

But where does this security originate? As you might have guessed, it stems from upstream restaking services such as EigenLayer, Symbiotic, and Karak.

Essentially, the security flow can be visualized as follows:



Verisense as a concise approach of pooled security

This model illustrates how security is conveyed from restaking services through the Verisense network to individual AVS, ultimately benefiting end applications and users.

Verisense provides an efficient and practical solution for shared security by acting as a high-performance, decentralized application container or cloud. It supports numerous AVSes with minimal operational costs.

However, this raises further questions:

- How is the subnet constructed? and what is the structure of an AVS subnet?
- How does the AVS subnet obtain pooled security from the Verisense hostnet?

To address these questions, Verisense introduces an innovative protocol named **Monadring**.

The Monadring Protocol

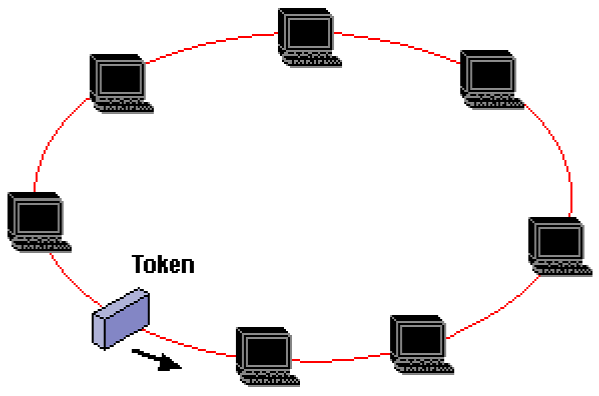
Monadring, serving as the backbone of the Validation-as-a-Service (VaaS) module, is a FHE-enabled token ring protocol that enables scalable and lightweight consensus within the Verisense network. Drawing inspiration from the token ring architecture, Monadring allows nodes within an existing blockchain network to form smaller, lightweight subnets. These subnets can perform computations more efficiently and cost-effectively, all while maintaining the robust security guarantees of the main blockchain.

To better understand Monadring, it's important to first grasp the concept of a token ring network.

What is Token Ring Network

A Token Ring network is a type of local area network (LAN) where devices are connected in a logical ring topology. This network design, developed by IBM in the early 1980s and standardized under IEEE 802.5, uses a token—a special data packet—that circulates around the network. Only the device holding the token can send data, a method that prevents collisions that might occur if multiple devices try to transmit simultaneously, as seen in architectures like Ethernet.

This token-based approach to managing access to the network shares similarities with the consensus mechanisms in blockchain networks, where a specific method determines which nodes can produce blocks.



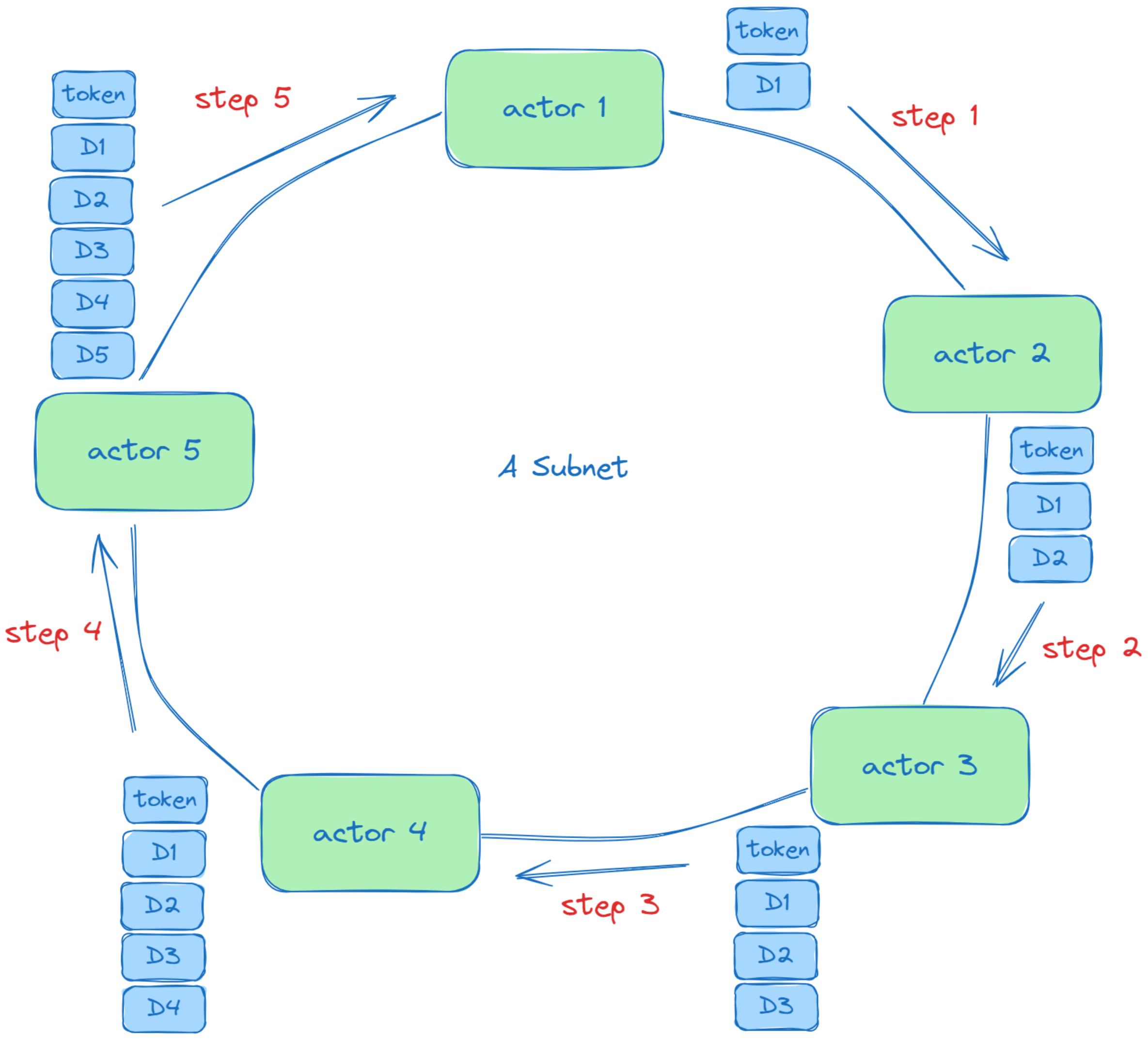
Token ring network

How Monadring Adopts Token Ring Design

As mentioned earlier, a token in a network functions like a boat carrying data to the next station. In Monadring, all involved actors are organized into a circular structure where a token is passed around. Each actor is a process running on a physical node, dedicated to one active AVS validation cluster. As the token circulates, it collects and carries data from each actor it encounters.

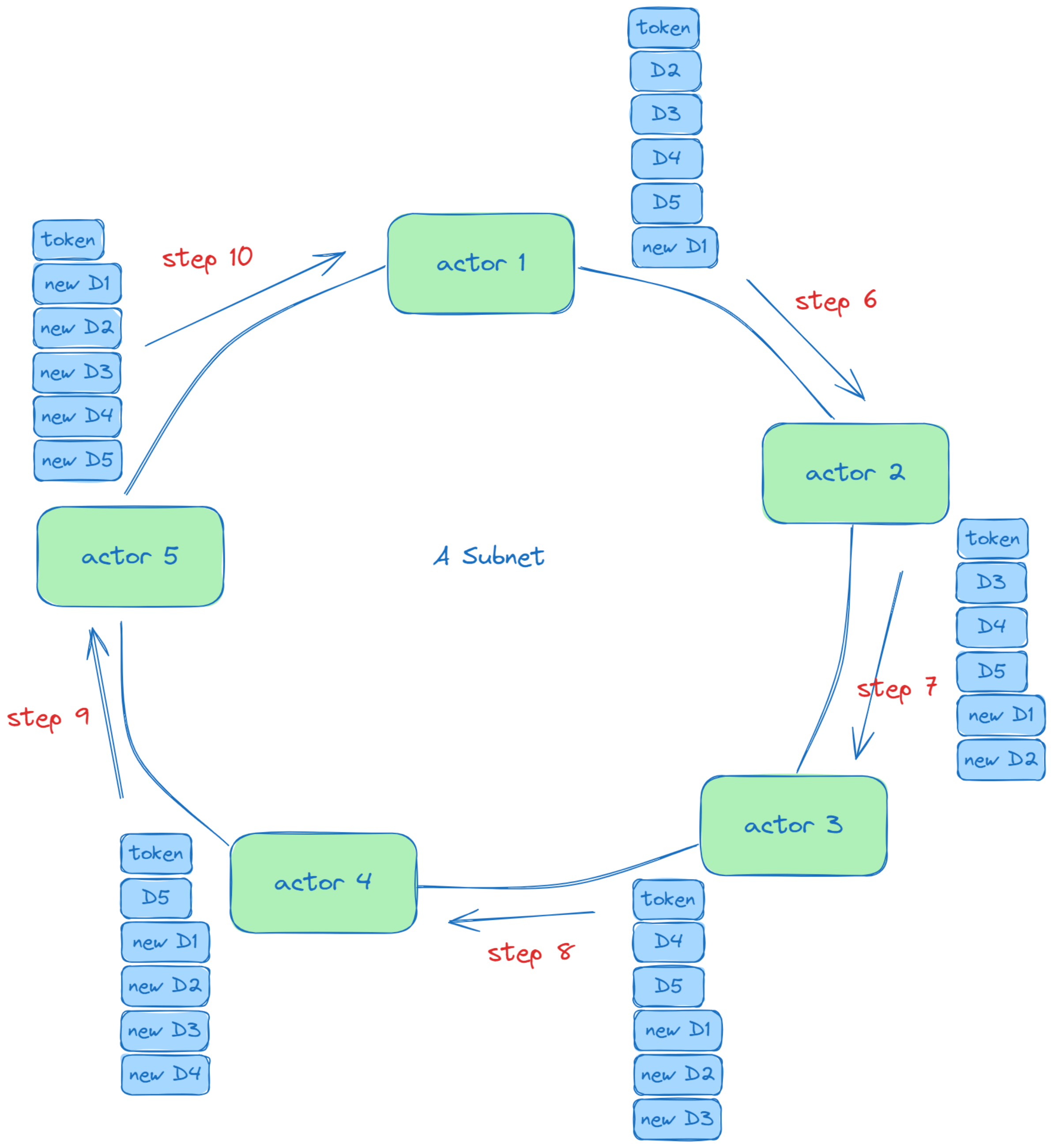
For example, when the token starts at Actor 1, Actor 1 executes its task and records as Data 1 (D1), and the token then carries D1 as it moves clockwise to Actor 2. Actor 2 will execute both D1 and D2 in sequence, attaching the D2 data package to the token before it continues to Actor 3. This process repeats for all actors in the subnet.

The diagram below illustrates the first round of circulation, detailing steps 1 through 5:



The first circle round of token and data

But what happens when the token completes its first circle and moves from the last actor (Actor 5) back to the first actor (Actor 1)? In the second round of circulation, something different occurs. The following diagram demonstrates the second round, covering steps 6 through 10:



The second circle round of token and data

As shown, when the token moves from Step 5 to Step 6, the payload of the token changes. The D1 package is consumed, and a new D1 package is attached to the token's data string. This deliberate design ensures that all actors in a subnet execute the same data package (including raw requests and directives) once, and update their local state (stored in a key-value database) to achieve final consistency across the subnet.

According to the Monadring Paper, the data packages like D1, D2, D3, etc., contain a list of modification events from pending request queue, the actor's digital signature, a digest of the actor's local ledger after applying the modifications, a q value indicating how many times this group should be delivered, the signer's `nonce`, and a `ct` value representing voting data using FHE (Fully Homomorphic Encryption).

$$(\mathbf{E} = [e_k, e_{k+1}..e_{k+n}], \mathcal{S}_{k+n}, \texttt{signature}, \texttt{nonce}, q, [\texttt{ct}, ..])$$

You don't need to grasp the full details of this formula right now, but it's important to understand that each data package contains crucial information.

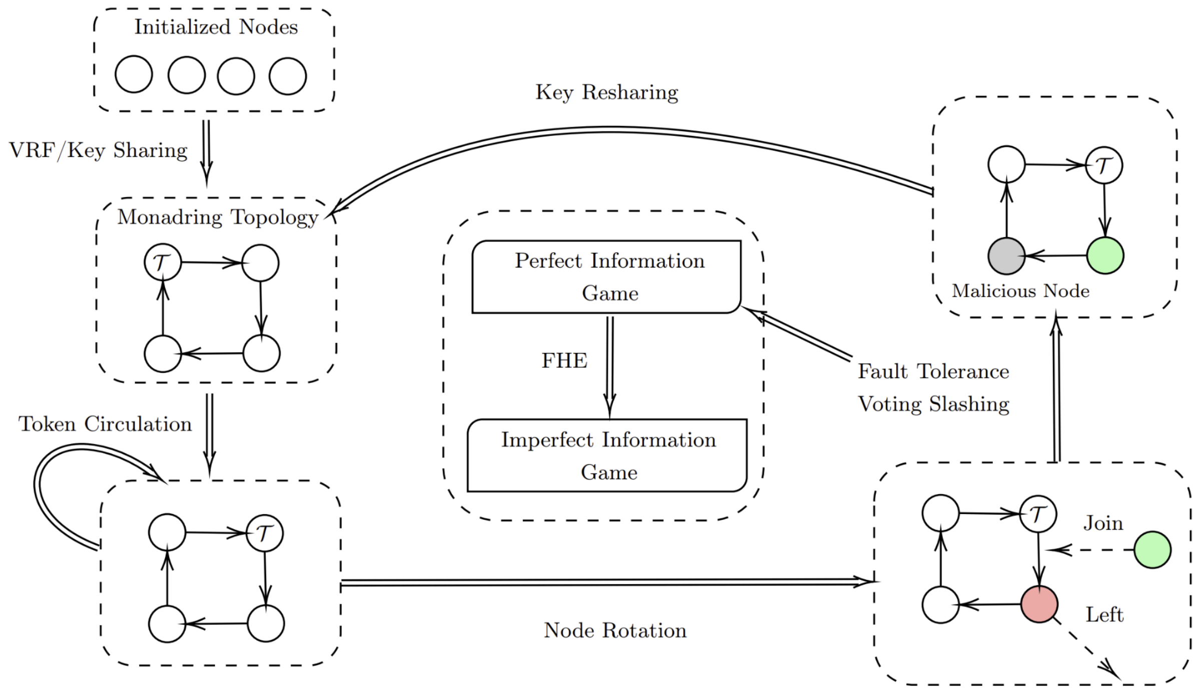
With this understanding of the Monadring subnet topology, we can further explore the dynamic mechanisms of Monadring.

The Mechanism of Monadring

According to the Monadring Paper, when a new AVS is registered on Verisense, Monadring initializes a set of nodes (actors) to join this AVS' subnet. There is a candidate node pool from which Monadring uses a Verifiable Random Function (VRF) to select nodes. These nodes are then configured as a token ring network, marking the start of the AVS subnet's circulation.

The concept of an epoch is central to this process. In blockchain, an epoch is a defined period during which specific activities occur within the network. At the beginning of each epoch, Monadring uses a randomness algorithm to rotate the node set of the AVS. This means some nodes (actors) are swapped out, and new nodes are invited in, creating a fresh validator set for the AVS subnet in the new epoch. This rotation enhances the security of the subnet.

This process is fundamental to Monadring's implementation of pooled security.



Monadring workflow

If a node detects any malicious behavior during the token ring circulation, it uses an FHE-enabled (Fully Homomorphic Encryption) algorithm to report the fraud. FHE ensures a blind voting thus the reporter's identity remains confidential to avoid the bribery yet the whole voting process can be traced as the proof of invalidity for the role of Resolver or Auditor with a private key, further enhancing the subnet's security.

FHE effectively transforms a perfect information game into an imperfect information game, a powerful and critical feature for Monadring's fraud tolerance and slashing mechanism.

The workflow can be summarized in the following pseudo code:

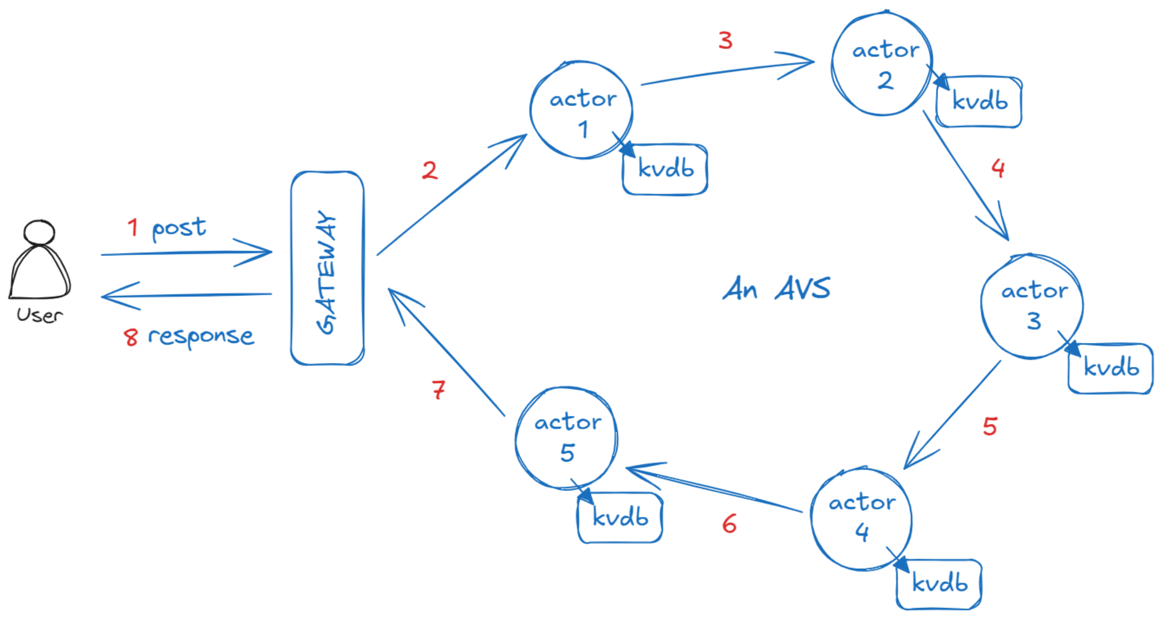
Algorithm 2 Monadring Workflow

```
1: Intialization: $sk, pk \leftarrow \texttt{FHEKeyGen}()$
2: Sends a transaction to the hostnet $\mathcal{V}$ to opt-in a subnet $\mathcal{S}_i$.
3: Ultilize VRF using Voting Model described in Section 2.4 to select the
   subnet members.
4: for each epoch do
5:     Sharing/Resharing the keys using Shamir's Secret Sharing Scheme.
6:     Compose a new group with initial $q \leftarrow n - 1$.
7:     while $q > 0$ do
8:         if receives $\mathcal{T}$ then
9:             Check the signature of each group $\mathcal{G}_i$.
10:            Apply the events of each group $\mathcal{G}_i$.
11:            Compare the digest with $\mathcal{L}_i$.
12:            Decrease the $q$ of executed groups.
13:            Handle transactions from the local queue.
14:            Deliver $\mathcal{T}$ to its successor.
15:        end if
16:        $q \leftarrow q - 1$
17:    end while
18:    Rotate the subnet members described in Section 3.2.1.
19:    if Malicious behaviours detected then
20:        ApplyFault tolerance with FHE described in Section 3.3.
21:        Voting for slashing using the voting model described in Sec-
   tion 2.4.
22:    end if
23: end for
```

For further details, please refer to the Monadring Paper.

The State Update Flow

How does an AVS process the user's input? The following diagram illustrates the entire procedure.



State update flow in an AVS

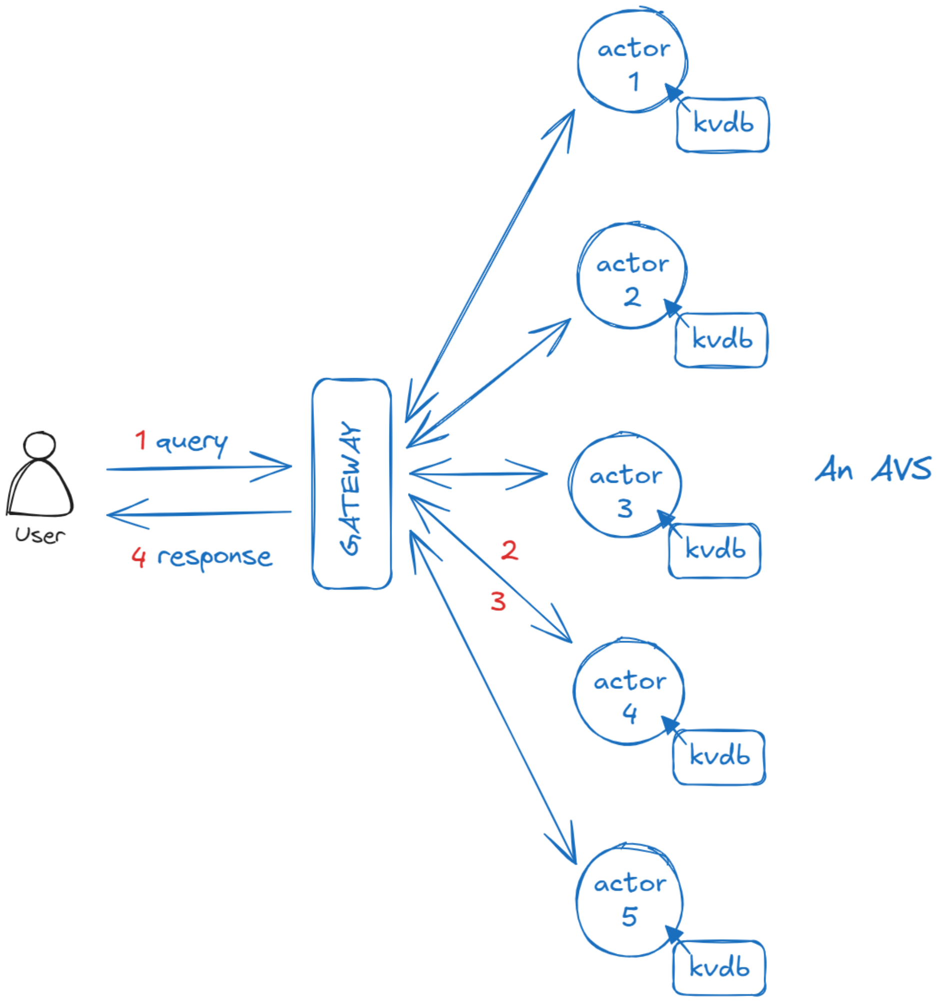
As described above, an AVS network is constructed as a token ring network. When a user makes a POST request to an AVS, the following flow is executed (the red numbers indicate the steps in the process):

1. The user initiates a POST request to the Verisense Gateway.
2. The Gateway routes this request to the targeted AVS subnet and activates one of the actors within this subnet (e.g., actor 1) to handle the request.
3. The selected actor processes the request, updates the corresponding state in its local key-value database (kvdb), and then forwards the request to the next actor in the subnet, along with a digest of its updated state.
4. The second actor receives the request and the digest from the first actor. It processes the request, updates its local kvdb, and calculates a digest of its updated state. This digest is compared with that of the first actor; under normal conditions, these digests should match.
5. This process repeats as the request circulates through the subnet (steps 4-6).
6. After completing a full cycle through the subnet, the response to the POST request is returned to the gateway.
7. Finally, the gateway forwards the response back to the end user.

As a result, each POST/update request traverses the entire AVS subnet before the response is returned to the user. By the end of this process, all actors within the AVS subnet will have synchronized and reached the same state.

The Query Flow

The query flow can be much simpler than the state update flow.



Query flow of an AVS

Let's go through the flow step by step as below.

1. Step 1: The process begins when a user sends a GET/query request to the Verisense Gateway.
2. Step 2: The Gateway then randomly routes the query to a specific actor within the target AVS subnet (for example, actor 4).
3. Step3: The selected actor processes the query, compiles the result, and sends it back to the Gateway.
4. Step 4: Finally, the Gateway delivers the response to the end user.

This streamlined flow ensures that the user receives the query response quickly, minimizing latency.

Fault Tolerance

Another critical component in the Monard ring is the design of the fault tolerance and the slashing detection. Verisense uses FHE to implement blind voting as a part of the proof of -validity.

Voting Game Model

In this chapter, we delve into the application of game theory within the Monadring system to design voting mechanisms that ensure the smooth operation of subnets. Understanding the mathematical principles behind various voting mechanisms is crucial for each AVS, as they must develop tailored voting strategies for their specific subnets.

From the perspective of voters, the goal is to maximize their individual benefits from voting. Conversely, the subnet's objective is to achieve network-wide consensus swiftly while minimizing reward distribution.

We begin by introducing fundamental game theory concepts and then apply these to the Monadring subnets.

For perfect information games, we present a theoretically optimal voting strategy where all participants have complete knowledge of each other's actions and the game's details. For imperfect information games, voters seek the optimal strategy (denoted as ξ) to maximize their expected payoffs in reaching a Nash equilibrium [11, 12]. From the AVS designers' viewpoint, the aim is to identify an appropriate threshold (θ) that minimizes overall payouts while maintaining effective consensus.

Voting Game

In this voting game, we consider a scenario with n voters, each assigned a voting weight ωi. Voters can cast an affirmative vote ($\top$) or a negative vote ($\perp$). The system sets a consensus threshold percentage θ, which determines the validity of the voting outcome.

The payoff function for each voter $b(\omega i; \theta_\top, \theta_\perp, n)$ operates on the following principles:

1. If the voter's choice aligns with the majority (i.e., reaches the threshold), they receive a positive payoff $\beta(\omega i)$.
2. If their vote goes against the majority decision, they incur a penalty $-\alpha(\omega i)$.
3. If neither side reaches the threshold, the payoff is 0.

We introduce deterministic information: when the i-th voter votes, they know there are already $n^{(i)}_\top$ affirmative votes and $n^{(i)}_\perp$ negative votes. This creates a unique "subgame" scenario where the threshold $\theta^{(i)}$ faced by each voter may vary based on the votes already cast.

In some cases, one side may have already secured enough votes to reach the threshold, making the decision clear. For example, if $n^{(i)}_\top \geq \theta_\top \cdot n$, then the affirmative votes have already reached the threshold, and θ^(i)ᵀ becomes 0. In other cases, the vote remains undecided, and voters must carefully consider their choices.

It's worth noting that although voters know the number of votes already cast, they don't know how the remaining n-n^(i)-1 participants will vote. This uncertainty adds a strategic layer to the decision-making process. Voters must balance their preferences, predictions about future votes, and potential payoffs or penalties.

This model aims to capture the complex dynamics of real-world voting systems, especially in distributed systems or blockchain networks, where reaching consensus among multiple parties with varying levels of influence is crucial. By introducing the weight ωi, we can simulate the different levels of influence of participants in the network, making the model closer to actual situations.

Perfect Information Scenarios

In this section, we explore a special voting scenario: the case of perfect information. In this situation, each voter knows the voting behavior of all previous voters. Specifically, we assume that the i-th voter has complete knowledge of the voting decisions of the preceding i-1 voters. This setting provides an ideal starting point for analyzing voter strategies.

Let's first consider the case of the last voter. Assuming all voters are rational, the strategy for the last voter becomes relatively straightforward. Their rational voting result v* can be expressed as:

1. Vote in favor if the number of affirmative votes plus their own vote exceeds the affirmative threshold.
2. Vote against if the number of negative votes plus their own vote exceeds the negative threshold
3. Otherwise, choose randomly

Here, v* represents the rational voting result, not the actual vote.

For other voters, the situation becomes more complex. Each voter needs to consider not only the known voting results but also predict the behavior of subsequent voters. Their rational voting result can be inferred based on the actual previous votes and expectations of future votes.

In an ideal scenario, if we assume that no voter has any additional biasing information, the voting result is no longer random. Instead, it can be determined based on the current voting situation and the set thresholds.

However, real situations are often much more complex than the ideal case. Especially for voters who are positioned earlier in the sequence, they face greater uncertainty. These voters need to estimate the behavior of more subsequent voters, which increases the difficulty of their decision-making. According to the Central Limit Theorem, the variance in this situation will be higher, reflecting the uncertainty in decision-making.

It's worth noting that for voters positioned before a specific point, decision-making becomes particularly challenging. This specific position is determined by the affirmative and negative thresholds as well as the current voting situation. This is because these voters need to estimate the behavior of a large number of subsequent voters, which is inherently uncertain.

Imperfect Information Scenarios

Imagine you're participating in a vote with n people, but you're the first to vote (i.e., n^(i) = 0), and you have no idea how others will vote. This is the scenario we're discussing.

In this situation, your vote is like playing a guessing game. We use ξ to represent the probability of each person voting in favor. This ξ is like a biased coin, where the value of ξ determines the probability of the coin landing heads up.

Now, let's consider the potential payoffs. We use b to represent the payoff function, and $\theta_\top$ and $\theta\perp$ to represent the thresholds for affirmative and negative votes, respectively.

1. If you vote in favor ($\top$), your expected payoff is:

   E[b | v_i = $\top$] = (Probability of winning the vote × Reward) - (Probability of losing the vote × Penalty)

2. Similarly, if you vote against ($\perp$), your expected payoff is:

   E[b | v_i = $\perp$] = (Probability of winning the vote × Reward) - (Probability of losing the vote × Penalty)

Our goal is to find the optimal value of ξ that maximizes the overall expected payoff. This overall expected payoff is:

$$
\begin{aligned}
&\mathbb{E}[b(\omega_i;\theta_\top,\theta_\perp,n)] \\
=&\mathbb{E}[b(\omega_i;\theta_\top,\theta_\perp,n)|v_i=\perp]\cdot(1-\xi)+\mathbb{E}[b(\omega_i;\theta_\top,\theta_\perp,n)|v_i=\top]\cdot\xi
\end{aligned}
$$

Through a series of complex calculations, we find an equilibrium point. At this point, slightly increasing or decreasing the value of ξ won't improve the expected payoff.

Interestingly, we find that this optimal ξ value equals:

$$\xi = \theta\perp / (\theta_\top + \theta\perp)$$

This means that the probability you should vote in favor equals the negative vote threshold divided by the sum of affirmative and negative vote thresholds.

We also discovered two special cases:

1. When $\xi = 0$ or $\xi = 1$, it's also an equilibrium state. This means if everyone votes in favor or everyone votes against, no one can improve their outcome by changing their vote.
2. When the reward and penalty are equal (denoted as $\alpha = \beta$), and $\theta_\top = \theta\perp$, the optimal strategy is $\xi = 0.5$, meaning it's like a fair coin toss with a 50% chance of voting in favor and a 50% chance of voting against.

This analysis helps us understand how people should rationally vote when they have no information. It also provides guidance for designing voting systems, telling us how to set $\theta_\top$ and $\theta\perp$ to get reasonable results even when information is lacking.

Transform Perfect Information Game to Imperfect Information Game with FHE

We propose a method to transform perfect information games into imperfect information games using Fully Homomorphic Encryption (FHE) techniques[6,7].

The procedure is as follows:

1. Key Generation:

   Each player creates a pair of keys (a secret key and a public key) using a specific encryption scheme called BFV [8, 9, 10].

2. Strategy Encryption:

   Players encrypt their strategies using their public keys. This means that each player's strategy is converted into a form that others can't understand without the corresponding secret key.

3. Encrypted Strategy Distribution:

   Players share their encrypted strategies with all other players. At this point, everyone has all the strategies, but they can't understand what those strategies actually are.

4. Homomorphic Payoff Calculation:

   Each player calculates their potential winnings (payoff) using the encrypted strategies. The special property of homomorphic encryption allows calculations to be performed on encrypted data without decrypting it first.

5. Payoff Decryption and Proof Generation:

   Finally, each player decrypts their own payoff. They also create a special mathematical proof that shows they calculated their payoff correctly, without revealing their secret key or their actual strategy.

The key idea is that this process allows players to participate in the game without knowing each other's actual strategies. They can calculate payoffs based on encrypted information, and prove they did so honestly, without revealing any secrets. This transforms what would normally be a perfect information game (where everyone knows all the information) into an imperfect information game (where some information remains hidden).

This method ensures fairness and privacy in the game, as players can verify the correctness of the results without seeing each other's private information. It's a sophisticated way of playing a game where you need to keep your moves secret but still prove you're following the rules.

Algorithm 1 Transform Perfect Information Game to Imperfect Information Game with FHE

```
Require: Number of players $n$, Security parameter $\lambda$
Ensure: Encrypted strategies $\{\sigma_i^{\texttt{Enc}}\}_{i=1}^n$, Payoffs $\{\pi_i\}_{i=1}^n$, Proofs $\{\Pi_i\}_{i=1}^n$
 1: for each player $i \in \{1, \ldots, n\}$ in parallel do
 2:     $\texttt{sk}_i \leftarrow \texttt{BFV.SecretKeyGen}(\lambda)$
 3:     $\texttt{pk}_i \leftarrow \texttt{BFV.PublicKeyGen}(\texttt{sk}_i)$
 4:     $\sigma_i^{\texttt{Enc}} \leftarrow \texttt{BFV.Enc}(\texttt{pk}_i, \sigma_i)$
 5:     Broadcast $\sigma_i^{\texttt{Enc}}$ to all other players
 6: end for
 7: Synchronization point: wait for all players to broadcast their encrypted
    strategies
 8: for each player $i \in \{1, \ldots, n\}$ in parallel do
 9:     Receive $\{\sigma_j^{\texttt{Enc}}\}_{j \neq i}$ from other players
10:     $\pi_i^{\texttt{Enc}} \leftarrow f_i(\sigma_1^{\texttt{Enc}}, \ldots, \sigma_n^{\texttt{Enc}})$        ▷ Homomorphic computation
11:     $\pi_i \leftarrow \texttt{BFV.Dec}(\texttt{sk}_i, \pi_i^{\texttt{Enc}})$
12:     $\Pi_i \leftarrow \texttt{Zk.Prove}(\texttt{sk}_i, \pi_i^{\texttt{Enc}}, \pi_i, \texttt{BFV.Dec})$
13: end for
14: return $\{\sigma_i^{\texttt{Enc}}\}_{i=1}^n$, $\{\pi_i\}_{i=1}^n$, $\{\Pi_i\}_{i=1}^n$
```

PIG to IIG

Slashing

With blind voting secured by Fully Homomorphic Encryption (FHE), an actor within the subnet is isolated from the voting information of other actors, ensuring complete privacy and integrity within the voting process. This design prevents any single actor from gaining insights into the decisions of others, thereby reducing the risk of collusion or manipulation. The raw voting data is securely stored in each actor's state space, preserving the integrity of the voting outcomes while maintaining the confidentiality of individual votes.

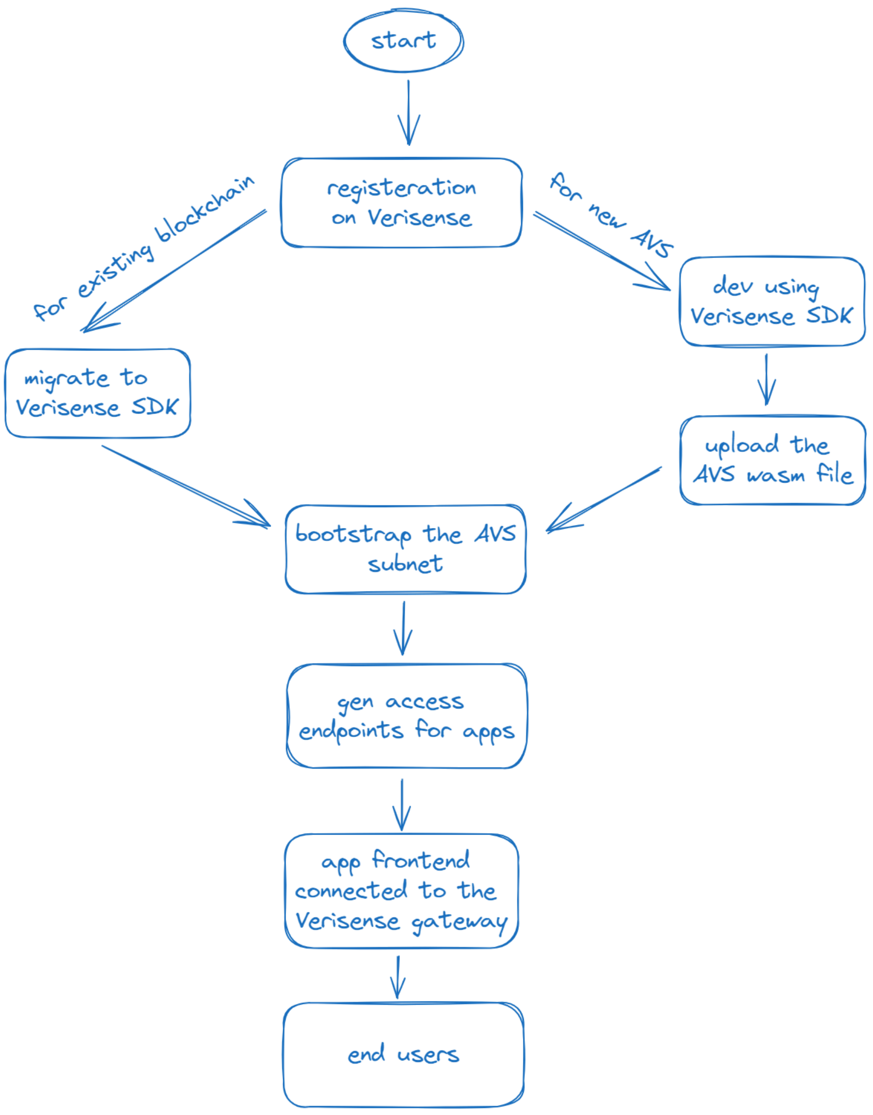
When it comes to the state roll-up process, the Verisense chain plays a crucial role in maintaining the network’s security and reliability. During this process, the state root of each actor's storage is uploaded to the Verisense chain, which then triggers a verification protocol. The runtime of the Verisense chain meticulously checks the signatures of all actors involved, ensuring that each signature is valid and corresponds to the correct actor. This verification step is vital for maintaining trust in the network and ensuring that no malicious activities go undetected.

Following the signature verification, the Verisense chain applies a simple majority rule to assess whether any actors have engaged in fraudulent or faulty behavior. This rule ensures that decisions are made democratically, reflecting the consensus of the majority while still allowing for the detection of anomalies. If the majority identifies discrepancies, such as conflicting states or unauthorized changes, these are flagged as potential fraud or faults. The simple majority rule serves as a critical mechanism for upholding the network’s integrity, ensuring that any dishonest behavior is swiftly identified.

Once fraud or faults are detected, the Verisense chain's runtime takes immediate action by reporting these issues to the upstream restaking providers, such as Symbiotic and EigenLayer. These providers are responsible for managing the slashing process, which has been predefined to handle such cases. The slashing process involves penalizing actors who have been found guilty of fraudulent activities or significant faults, typically by reducing their stake or imposing other penalties. This process not only deters malicious behavior but also reinforces the network's security by removing or penalizing bad actors.

In this architecture, the Verisense chain functions as a supervisory authority, overseeing the operations of the AVS subnets. It acts as a vigilant guardian, constantly monitoring for inconsistencies or signs of misconduct and ensuring that the rules of the network are enforced without bias. The supervision provided by the Verisense chain is essential for maintaining a secure and reliable environment for AVS applications.

How to Deploy an AVS on Verisense



Onboarding Verisense as an AVS

Deploying on Verisense is designed to be both simple and efficient, making it an attractive option for developers and organizations alike. Whether you're working with an existing blockchain or creating a new AVS from scratch, Verisense provides the tools and infrastructure to get you up and running quickly.

For Existing Blockchains

If you have an existing blockchain that you want to migrate to the Verisense ecosystem, the process is straightforward. Begin by registering your blockchain on the Verisense VaaS (Verisense-as-a-Service) platform. Once registered, you can easily migrate your blockchain to the Verisense SDK. The Verisense platform will then automatically bootstrap a new AVS subnet specifically for your blockchain. This seamless transition allows your blockchain to integrate with the broader Verisense network, leveraging its robust infrastructure and features.

For New AVS Development

If you’re starting from scratch and wish to build a brand new AVS, the process is equally intuitive. First, you’ll need to register on the Verisense VaaS platform. After registration, you can begin development using the Verisense SDK, which provides a familiar environment similar to developing a web backend service. This means that developers with web development experience can quickly adapt to the Verisense SDK, reducing the learning curve.

Once you’ve completed the development of your AVS, the next step is to compile your code into a WebAssembly (WASM) file. WebAssembly is a powerful format that ensures your AVS can run efficiently across different environments. After compiling your code, upload the WASM file to the Verisense VaaS platform. Upon upload, Verisense will automatically bootstrap a new AVS subnet, providing a dedicated and secure environment for your service.

Access and Integration

After the AVS subnet is bootstrapped, the Verisense VaaS platform will generate uniform endpoints that your app clients can use to access the AVS services. These endpoints are designed to be consistent and reliable, ensuring smooth communication between your frontend app and the AVS. You can then use your own frontend application to connect to the Verisense Gateway, which acts as the entry point for accessing the AVS services. This integration allows you to leverage the power of Verisense while maintaining full control over your application's user interface and client-side logic.

Whether you're migrating an existing blockchain or building a new AVS from the ground up, Verisense simplifies the deployment process, providing a robust and scalable platform for your needs. With its automatic bootstrapping, uniform endpoints, and seamless integration capabilities, Verisense empowers developers to focus on innovation without worrying about the complexities of deployment and infrastructure management.

Key Features and Benefits of Verisense

After detailing the mechanisms of Verisense’s VaaS, Monadring, and the blind voting design, we can now highlight the key features that make Verisense a valuable asset for AVSes and restaking partners.

1. **Scalable Consensus Mechanism**: Verisense offers a highly flexible and adaptable consensus mechanism tailored to the specific needs of various AVS applications. This allows Verisense to form a lightweight consensus with the desirable degree of

decentralization and customizable network size according to the AVS's security needs out of their business nature and the affordability. This scalability is essential for onboarding a diverse range of AVS clients.
2. **Balanced Speed and Security**: In addition to scalability, Verisense strikes a critical balance between speed and security, two often conflicting factors in consensus mechanisms. By leveraging the Token Ring Protocol for speed and implementing node swapping with VRF for enhanced security, Verisense delivers both rapid performance and robust protection.
3. **FHE-Enabled Security**: Fully Homomorphic Encryption (FHE) technology allows computations on encrypted data as if it were plain data, thus elevating security in blockchain systems. FHE enablement at the Verisense VaaS module (node level) transforms a perfect information game (i.e., a typical blockchain due to data transparency) into an imperfect information game (i.e., wrapping the voter identity and the vote outcome to avoid bribery), helping to reach and maintain Nash equilibrium of the entire network. This is particularly useful to apply the blind voting among participating nodes in an active AVS cluster as part of the proof of invalidity mechanism to detect malicious behaviors from nodes.
4. **Adaptive Slashing Mechanism**: Verisense introduces a flexible slashing mechanism with proof of invalidity, ensuring a fair and resilient method to monitor, detect, and report malicious behavior. Verisense does not initiate the execution of slashing by design, maintaining flexibility to integrate with any restaking layer and align with their slashing rules.
5. **Decoupled Consensus and Runtime:** By separating consensus from runtime at node level, Verisense empowers to support AVS applications that demand higher performance from the underlying blockchain, making it a versatile solution for diverse web2 use cases meeting web3 decentralization. This is a bold strid to explore the possibility of maximally releasing the node resources for extra yet crucial functions a blockchain normally fails to offer such as active data i/o calling, high frencent message passing, super affordable file searching and retrieving, and etc.
6. **Scientific Node Management:** Verisense employs a sophisticated node management system, optimizing node allocation to better serve AVS. High-performance and specialized nodes, such as those with hardware acceleration or legal permissions, gain a higher chance to serve matching AVS, ensuring higher and appropriate rewards.

Conclusion

Verisense presents a novel Validation-as-a-Service (VaaS) framework, offering a scalable, secure, and flexible solution for AVS applications within the restaking ecosystem. Through innovations like the Monadring protocol, blind voting design, and the integration of Fully Homomorphic Encryption (FHE), Verisense addresses key challenges in decentralized networks.

As we continue to refine Verisense, future updates will provide more implementation details and comparative analysis of the Monadring protocol against Byzantine Fault Tolerance (BFT), further demonstrating its superiority in balancing speed, security, and scalability.

Verisense stands poised to become a cornerstone in the evolution of decentralized validation services.

Reference

[1] Monadring paper.
https://github.com/verisense-network/papers/blob/main/monadring/monadring.pdf

[2] Understanding the EigenLayer AVS Landscape.
https://www.coinbase.com/en-ca/blog/eigenlayer

[3] EigenLayer Architecture.
https://docs.eigenlayer.xyz/assets/files/EigenLayer_WhitePaper-88c47923ca0319870c611decd6e562ad.pdf

[4] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. "A survey on homomorphic encryption schemes: Theory and implementation." ACM Computing Surveys (CSUR) 51, no. 4 (2018): 1-35.

[5] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. "TFHE: fast fully homomorphic encryption over the torus." Journal of Cryptology 33, no. 1 (2020): 34-91.

[6] Zvika Brakerski. "Fully homomorphic encryption without modulus switching from classical GapSVP." In Advances in Cryptology – CRYPTO 2012, edited by Reihaneh Safavi-Naini and Ran Canetti, 868-886. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[7] Junfeng Fan and Frederik Vercauteren. "Somewhat practical fully homomorphic encryption." Cryptology ePrint Archive, Paper 2012/144, 2012. https://eprint.iacr.org/2012/144.

[8] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. "(Leveled) fully homomorphic encryption without bootstrapping." ACM Transactions on Computation Theory (TOCT) 6, no. 3 (2014): 1-36.

[9] Drew Fudenberg and Jean Tirole. Game theory. MIT press, 1991.

[10] Constantinos Daskalakis, Paul W Goldberg, and Christos H Papadimitriou. "The complexity of computing a Nash equilibrium." Communications of the ACM 52, no. 2 (2009): 89-97.

[11] Castro M, Liskov B. Practical byzantine fault tolerance[C]//OsDI. 1999, 99(1999): 173-186.

[12] King S, Nadal S. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake[J]. self-published paper, August, 2012, 19(1).