Monadring: A lightweight consensus protocol for organizing subnets upon a blockchain system

Zhang Yu[*], Yan Xiao[*], Gang Tang[*], and Helena Wang[*]

[*]Verisense

June 15, 2024

Abstract

Existing blockchain networks are often large-scale, requiring transactions to be synchronized across the entire network to reach consensus, and on-chain computations can be prohibitively expensive, making many CPU-sensitive computations infeasible.

Inspired by the structure of IBM's token ring networks, we propose a lightweight consensus protocol called Monadring to address these issues. Monadring allows nodes that are already part of a large blockchain network to form a smaller subnetwork to perform computations more quickly and cheaply, while still maintaining the same security guarantees as the main blockchain network.

To further enhance the security of Monadring, we introduce a node rotation mechanism based on Fully Homomorphic Encryption (FHE) within the smaller subnetwork. Unlike the common voting-based election of validator nodes, the election in Monadring leverages FHE to hide the voting information, eliminating the advantage of the last mover in the voting game.

The paper details the design and implementation of the Monadring protocol, and evaluates its performance and feasibility through simulation experiments. This research contributes to enhancing the practical utility of blockchain technology in large-scale application scenarios.

1 Introduction

Recent blockchain systems have often adopted faster and more energy-efficient consensus protocols like Ouroboros, BABE, and Tendermint. Blockchain networks are typically open and permissionless, and we can define the degree of decentralization of such networks by the number of participating nodes. Leveraging the consensus protocols mentioned earlier, it is possible to rapidly construct new blockchain networks. However, decentralized applications often cannot flexibly customize their degree of decentralization based on the importance of their underlying data.

For example, decentralized social media applications may require faster response times and lower storage costs compared to decentralized finance applications, and hence may need a lower degree of decentralization. This flexibility to adjust the level of decentralization based on the needs of the application is an important consideration that has not been fully addressed in existing blockchain architectures.

Building a subnetwork over an existing blockchain system is a possible way to address this issue. By drawing inspiration from the token ring architecture, we propose a novel consensus protocol called Monadring that aims to enable nodes within an existing blockchain network to form smaller, lightweight subnetworks capable of performing computations more efficiently and cost-effectively, while still maintaining the same security guarantees as the main blockchain.

Token ring network operates at the data link layer of the OSI/RM model. It was designed to solve the problem of physical link contention just like Ethernet. In a token ring network, a token is passed sequentially from one node to the next, granting the holder the right to transmit data. This token-based system for managing access to the shared medium bears some similarity to the consensus mechanisms used in blockchain networks for selecting block producer.

Despite the fact that token ring networks have largely fallen out of favor in modern networking due to their limited scalability and other drawbacks, the underlying principles of their decentralized, token-based structure could provide valuable insights for designing a lightweight consensus algorithm for small blockchain systems.

Monadring also involves FHE-based member rotation mechanism to enhance the security. Homomorphic encryption (HE) is a method of encryption that allows computations to be carried out on encrypted data, generating an encrypted result which, when decrypted, matches the outcome of computations performed on the plaintext. This property enables sophisticated computations on encrypted data while maintaining data security. HE schemes are a type of encryption method that can protect data privacy because they allow computations to be performed directly on the encrypted data. For example, an HE scheme might allow a user to perform operations like addition and multiplication on encrypted numbers, and these operations would have the same result as if they were performed on the original, unencrypted numbers. This technology is seen as a key component for secure cloud computing since it allows complex data manipulations to be carried out on completely secure encrypted data.

Fully Homomorphic Encryption (FHE) is a more advanced form of Homomorphic Encryption. FHE allows arbitrary computations to be carried out on encrypted data, which is not the case with normal HE which might be limited in the types of computation it supports. FHE computations generate a result that, when decrypted, corresponds to the result of the same computations performed on the plaintext. This makes FHE extremely useful for cases where sensitive data must be processed or analyzed, but security and privacy considerations prevent the data from being decrypted. With FHE, you can perform unlimited calculations on this encrypted data just like you would on unencrypted data. For instance, in the field of cloud computing, FHE allows users to operate computations on encrypted data stored in the cloud, preserving data confidentiality and privacy.

2 Definitions and Model

2.1 FHE

Some of the more popular FHE frameworks include BFV, BGV and CKKS.

- **BGV (Brakerski-Gentry-Vaikuntanathan)**: The BGV scheme is a Fully Homomorphic Encryption (FHE) method, proposed by Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. It offers a choice of FHE schemes based on the learning with error (LWE) or ring-LWE problems that have substantial security against known attacks. BGV allows the encryption of a single bit at a time and the efficiency of the encryption is largely considered in cloud storage models.

- **BFV (Brakerski/Fan-Vercauteren)**: BFV is another homomorphic encryption scheme that is often considered for its practical performance alongside the BGV scheme. BFV supports a set of mathematical operations such as addition and multiplication to be performed directly on the encrypted data. It has been implemented efficiently and there have also been several optimizations proposed to enhance its performance in different applications.

- **CKKS (Cheon-Kim-Kim-Song)**: The CKKS scheme is known for being a Leveled Homomorphic Encryption method that supports approximate arithmetic operations over encrypted data. The CKKS scheme is especially suitable for computations involving real or complex numbers. Its ability to perform operations on encrypted data without the necessity for decryption makes it highly useful for maintaining data security during computations.

The Brakerski-Gentry-Vaikuntanathan (BGV) and Brakerski/Fan-Vercauteren (BFV) schemes differ mainly in how they encode information. BGV encodes messages in the least significant digit (LSD) of the integer, while BFV encodes messages in the most significant digit (MSD) of the integer. This difference can affect how the encrypted data is handled and manipulated during computations.

Mathematic notations of FHE. We define the symbol $\mathbb{Z}$ as the set of integers, $\mathbb{Q}$ as the field of rational numbers, $\mathbb{R}$ as the field of real numbers, and $\mathbb{C}$ as the field of complex numbers.

Further, we define the polynomial cyclotomic ring $\mathcal{R}$ as follows:

$$\mathcal{R} := \mathbb{Z}[X]/(X^N + 1)$$

Correspondingly, $\mathcal{R}_q$ is defined as $(\mathbb{Z}/q\mathbb{Z})[X]/(X^N+1)$, where typically $N = 2^n$, that is, N is an integer power of 2.

We define a distribution as $\chi_{\sigma^2,\mu}$, where σ^2 denotes the variance and μ signifies the mean. The uniform ternary distribution and discrete Gaussian distribution are represented by $\chi_{\mathcal{T}}$ and $\chi_{\mathcal{N}}$ respectively. Considering the standard BFV encryption formula:

$$\mathbf{a} \cdot \mathbf{s} + \Delta m + \mathbf{e}$$

It's established that $\mathbf{a} \sim \chi_{\mathcal{T}}$ and $\mathbf{e} \sim \chi_{\mathcal{N}}$. Wherein, the probability density function of the discrete Gaussian distribution is represented as $\rho_{\mu,\sigma^2} = e^{-\|\mathbf{x}-\mu\|^2/2\sigma^2}$.

The expansion factor $\delta_{\mathcal{R}}$ is defined as $\|\mathbf{a} \cdot \mathbf{b}\|_\infty / (\|\mathbf{a}\|_\infty \cdot \|\mathbf{b}\|_\infty)$ where $\mathbf{a}, \mathbf{b} \in \mathcal{R}$

Shamir Secret Sharing Scheme. Shamir's Secret Sharing is an algorithm in cryptography devised by Adi Shamir. It's a form of secret sharing, where a secret is divided into parts, giving each participant its own unique part. The unique feature of the algorithm is the minimal amount of parts, or shares, needed to reconstruct the secret. Here's a simple walkthrough of how Shamir's Secret Sharing can be used for threshold private key sharing:

1. Choose the Threshold: Define the threshold number t below which knowing t points gives no information about the secret, but t points yields the secret.

2. Generate a Polynomial: Generate a random polynomial of degree $t-1$ with the constant term being the secret (private key) to be shared. i.e. .

$$\mathcal{P}(\mathbf{x}) = a_0 + a_1\mathbf{x} + a_2\mathbf{x}^2 + \ldots + a_{t-1}\mathbf{x}^{t-1} \tag{1}$$

3. Create Shares: Evaluate the polynomial at different points to get n shares, where n is the total number of participants. Each participant is given one share, which is a point on the polynomial. i.e.

$$s_i = \mathcal{P}(\mathbf{x}_i) \tag{2}$$

4. Distribute the Shares: The shares of the private key are then distributed among the participants. The key property here is that any t shares (points) are enough to reconstruct the polynomial (and hence discover the secret), whereas $t-1$ or fewer shares reveal no information about the secret.

5. Reconstruct the Secret: When the need arises to use the private key (secret), any t participants come together and combine their shares using polynomial interpolation (for example, via Lagrange interpolation) to reconstruct the polynomial and discover the constant term, which is the secret.

$$\mathcal{P}(x) = \sum_{i=0}^{t-1} s_i \prod_{j\neq i} \frac{x - s_j}{s_i - s_j} \tag{3}$$

It is worth noting that all polynomials are defined over the ring of polynomials in $(\mathbb{Z}/p\mathbb{Z})[X]/X^t$ and the Lagrange interpolation still holds.

This way, the private key (secret) is never explicitly revealed to any single party and no single party can access the secret alone. This is particularly useful in managing the risks associated with key management in cryptographic systems. It provides a balance between accessibility (through the threshold number of participants) and security (no single point of failure).

BFV Scheme. As described in the citation [**?**, **?**], the BFV scheme proposes the method of placing the ciphertext in the Most Significant Digit (MSD) position, which makes the trend of noise growth change from quadratic to linear. This not only greatly reduces the influence of noise in the computation process, but also eliminates the need for modulus switching, thus effectively enhancing the computational efficiency.

- $\texttt{BFV.SecretKeyGen(}1^\lambda\texttt{)}$: Generate a secret key $\texttt{sk} \leftarrow \chi_{\mathcal{T}}$.
- $\texttt{BFV.PublicKeyGen(sk)}$: Generate a public key $\texttt{pk} = (\texttt{pk}_0, \texttt{pk}_1) = ([-\mathbf{a} \cdot \mathbf{s} + \mathbf{e}]_q, \mathbf{a})$ where $\mathbf{e} \leftarrow \chi_{\mathcal{N}}$ and $\mathbf{a} \leftarrow \chi_{\mathcal{T}}$.
- $\texttt{BFV.Enc(pk,m)}$: Message $\mathbf{m} \in \mathcal{R}_p$, where $p < q$. $\mathbf{u} \leftarrow \chi_{\mathcal{T}}$ and $\mathbf{e}_0, \mathbf{e}_1 \leftarrow \chi_{\mathcal{N}}$. The ciphertext is

$$\texttt{ct} = (\texttt{ct}_0, \texttt{ct}_1) = ([\texttt{pk}_0 \cdot \mathbf{u} + \Delta \cdot \mathbf{m} + \mathbf{e}_0]_q, [\texttt{pk}_1 \cdot \mathbf{u} + \mathbf{e}_1]_q)$$

- $\texttt{BFV.Dec(sk,ct)}$:

$$\mathbf{m} = [\lfloor p \cdot [\texttt{ct}_0 + \texttt{ct}_1 \cdot \mathbf{s}]_q / q]]_t$$

It is worth noting that the BFV scheme differs from the BGV scheme in that the BGV ciphertext is placed in LSD, i.e.

- $\texttt{BGV.PublicKeyGen(sk)}$:

$$\texttt{pk} = ([-\mathbf{a} \cdot s + p\mathbf{e}]_q, \mathbf{a})$$

- $\texttt{BGV.Enc(pk,m)}$:

$$\texttt{ct} = ([\mathbf{m} + \mathbf{u} \cdot \texttt{pk}_0 + p\mathbf{e}_0]_q, [\mathbf{u} \cdot \texttt{pk}_1 + p\mathbf{e}_1]_q)$$

We need to evaluate the effect of the encryption function $\texttt{Enc}$ and the decryption function $\texttt{Dec}$ on noise. Consider the following equation:

$$\texttt{ct}_0 + \texttt{ct}_1 \cdot s = \Delta\mathbf{m} + p\mathbf{v} \tag{4}$$

The noise $\mathbf{v}$ can be evaluated by $\|\mathbf{v}\|_\infty < (q - pr_t(q))/(2p)$ where $r_p(q) = q - p\Delta$ which is a value determined by the modulus q, the plaintext scaling factor Δ and the noise ratio p.

This condition exists because during the decryption process we actually perform a p/q scaling operation. This scaling operation may amplify the effect of the noise, so we need to make sure that the size of the noise is within an acceptable range before decryption.

2.2 Network Architecture

Hostnet and subnet. Consider a network composed of a set of participants $\mathbb{V}$ in which the majority obey a protocol to reach Byzantine Agreement and finality[?] over ledger $\mathcal{L}$. S_i is a subset of $\mathbb{V}$, $\mathbb{V} = S_0 \cup S_1 \cup .. \cup S_{n-1}$. We call $\mathbb{V}$ is a *hostnet* and S_i is a *subnet* of $\mathbb{V}$.

- A node participant $v \in \mathbb{V}$ could be a member of any S_i at meantime, $v \in \mathbb{V}, v \in S_m \cap S_n$ is valid.
- We assume the ledger $\mathcal{L}$ of hostnet maintains all the subnets information in form of a mapping *subnet id→node list.*
- Any participant of the hostnet could join a specific subnet through proposing a modification over ledger $\mathcal{L}$.
- Since the ledger $\mathcal{L}$ is under Byzantine Agreement by all participants, the map can be considered a provable information outside any subnets.

Subnet ledger. For each subnet S_i, all participants $v_j \in S_i$ maintain an independent ledger L_i different from the ledger $\mathcal{L}$ of hostnet, while the root state of each subnet ledger will be recorded in the hostnet ledger, $L_i \nsubseteq \mathcal{L}, \mathcal{F}(L_i) \in \mathcal{L}$.

A subnet could handle query and update requests independently from the hostnet ledger.

We want to formalise the procedure of reaching consensus on the ledger across all participants as a protocol that can be deployed along with any kind of blockchain network to build subnet. We can assume that the subnet ledger has properties as below:

- The subnet ledger contains all the modification events and each event has an increasemental number as index. The ledger is expected to be in a deterministic state after the n_{th} event being applied, $S_{n+1} = f(S_n, e_n)$.
- Particularly, changing the function f is also a kind of event. The first event e_0 is loading the function.
- The time complexity of looking up the n_{th} event is $O(1)$.
- The time complexity of retrieving the maximum event id is $O(1)$.

Subnet topology and token. The participants of the subnet strictly follows the sequence of the `node list` to form a ring topology. A `token` $\mathcal{T}$ of a subnet with n nodes is a special signal circulates around the ring following the sequence of the `node list`.

- The token carries groups of events from its sender and the sender's forehead recursively. Group G_i is composed by the node v_i. It contains a list of modification events originally from its pending request queue, the node's digital signature, a digest of its local ledger after these modification applied, a number q indicates how many times this group should delivered and the nonce of the signer:

  $$G_i = (\mathbf{E} = [e_k, e_{k+1}..e_{k+n}], S_{k+n}, \texttt{sig}(\texttt{nonce}, \mathbf{E}, S_{k+n}), \texttt{nonce}, q) \quad (5)$$

  The event list $\mathbf{E}$ could be empty, the q could be negative.

- Normally, a token circulates in a subnet with n nodes should always include n groups unless there were malicious behaviors or some nodes went offline. Whenever a node receives the token, it ought to check the signature for each group. Then applying all the events of each group and compare the digest with the local ledger. The q of executed groups should be decreased by 1.

- If all checks pass, the node should handle transactions from its local queue and compose them as a new group with initial $q = n - 1$ to replace previous one in the token. Then try to deliver the token to its successor.

Assume a subnet with n nodes $v_0, v_1..v_{n-1}, n \geq 3$. If no nodes joined the subnet during last round, when a node v_i receives the token $\mathcal{T}$, noted as T_i.

$$\begin{aligned} T_i &= G_i, G_{(i+1) \mod n}, .., G_{(i+n-1) \mod n} && (6) \\ T_{i+1} &= G_{i+1}, G_{(i+2) \mod n}, .., G'_i; i+1 < n && (7) \end{aligned}$$

Launching a subnet and rotating subnet members.

Function upgrade. Function upgrade is a kind of event which should be included in the token G either. Only some specific users could initiate this event, wherein the subnet manager submit a transaction to modify the hostnet ledger $\mathcal{L}$ similar to registering a subnet.

Besides the function itself, a digest and version should be included in this transaction. E.g. An executable WASM binary with digest and version.

Thus, once a upgrading transaction is confirmed by hostnet, any members of the subnet could include this event into token $\mathcal{T}$ if the hostnet ledger contains a higher version function but an associated event is absent in the token.

Fault tolerence. The hostnet can reach the Byzantine Agreement over L based on an assumption that the ratio of honest participants $r >$ `threshold`. We don't expect that a subnet has a same ratio. We neither don't expect that a node is honest on ledger L would be honest on subnet legder L_i. We want to find a solution to detect the malicious behaviours or nodes offline then exclude them from the subnet by a provable invalidity.

For simplicity, we note $T_{i/i}$ as T_i excluding the i_{th} group. An valid T_i must satisfy properties as below:

- The q of G_i satisfies $q \leq 0$. $q = 0$ implies no new nodes joined the subnet in this round, while $q < 0$ implies some new nodes have joined the subnet. The node v_i can check the *node list* from L. For rest q of $\forall G \in T_{i/i}$ are similar.
- TODO check signatures.
- Assume the maximum event id of G_i is m. Combine all event lists of $T_{i/i}$ as a single list $E_{i/i}$. Assume the minimal event id of $E_{i/i}$ is n. $n = m + 1$ and m is the maximum event id of node v_i.

3 The Monadring Protocol