Copyright ⓒ 2022 KSII

Surveillant: a supervision mechanism between blockchains for efficient cross-chain verification

Xinyu Liang[1*], Jing Chen[2], Ruiying Du[3], and Tianrui Zhao[4]
[1 2 3 4] School of Cyber Science and Engineering, Wuhan University
Wuhan, 430072 China
[1] [e-mail: liangxinyu@whu.edu.cn]
[2] [e-mail: chenjing@whu.edu.cn]
[3] [e-mail: duraying@126.com]
[4] [e-mail: zhaotianrui@whu.edu.cn]
[*] Corresponding author: Xinyu Liang

Received December 23, 2021; revised February 12, 2022; revised June 1, 2022; accepted July 8, 2022; published August 31, 2022

Abstract

Blockchain interoperability, which refers in particular to the ability to access information across blockchain systems, plays the key role for different blockchains to communicate with each other, and further supports the superstructure built on top of the cross-chain mechanism. Nowadays, blockchain interoperability technology is still in its infancy. The existing cross-chain scheme such as BTCRelay requires that the smart contract in a blockchain to download and maintain block headers of the other blockchain, which is costly in maintenance and inefficient to use. In this paper, we propose a supervision mechanism between blockchains, called Surveillant. Specially, the new entities called dual-functional nodes are introduced to commit the real-time information from the blockchain under supervision to the supervising blockchain, which enables users to have efficient cross-chain verification. Furthermore, we introduce Merkle mountain range for blocks aggregation to deal with the large-scale committing data. We propose the design of long orphan branch counter to trace the bifurcations in the blockchain under supervision. The existing incentive mechanism is improved to encourage the behaviors of dual-functional nodes. In Surveillant, the analysis and experimental results demonstrate that users are able to have efficient cross-chain verification with low maintenance overhead.

Keywords: blockchain interoperability, cross-chain, BTCRelay.

This research of Wuhan University was supported in part by the National Key R&D Program of China under grant No. 2021YFB2700200, the Fundamental Research Funds for the Central Universities under grants No. 2042022kf1195, 2042022kf0046, and the National Natural Science Foundation of China under grants No. U1836202, 62076187, 62172303

http://doi.org/10.3837/tiis.2022.08.003 **ISSN : 1976-7277**

1. Introduction

In recent years, we have witnessed the rapid development of blockchain technology. Researchers have made considerable effort to improve the performance of blockchain, such as more efficient consensus protocols [1], better privacy protection [2], improving throughput by state channels [3] and sharding protocols [4]. Based on the works above, Blockchain has been widely used in various fields, such as the payment platform [5], smart contract [6], privacy preserving [7], etc.

Although the number and variety of blockchain applications are increasing, these applications remain mutually isolated, in which each blockchain only operates within its own ecosystem. Subsequently, a new demand of achieving *blockchain interoperability* [8] has naturally emerged. It requires to establish a secure and practical cross-chain mechanism to transfer information across blockchains. By this mechanism, different blockchain systems begin to communicate with one another to form a blockchain web, which is invaluable for next generation blockchain technology [9].

Illustrated by Vitalik Buterin [9], the existing research on blockchain interoperability can be classified into three categories, which are notary scheme, hash-locking, and chain relay. First, in the notary schemes [10], the information is transferred between blockchains by a trusted third party, which has potential security vulnerability. Second, the hash-locking [11] is adopted to ensure the atomicity [12] of the operations on different blockchains. However, in this scheme, the information transferred between blockchains is very limited, which cannot support complex cross-chain operations. Third, in the chain relay schemes [13] [14], the smart contract in the first blockchain will receive and maintain the SPV proof of the second blockchain delivered by a group of relayers. It enables programmatic public cross-chain verification, which has the advantages in security and functionality compared with the notary scheme and hash-locking.

However, the existing chain relay schemes still suffer from several drawbacks as follows. First, as the chain relay schemes require to maintain the SPV proof of the other blockchain, it is comparable to that the smart contract additionally maintain a light client of the other blockchain, which is costly in maintenance and inefficient to use. Second, as a blockchain includes thousands or millions of blocks, how to commit the large-scale blocks across blockchains becomes a problem. Moreover, because blockchain has a dynamic changing data structure, a block having been included in the blockchain before has the possibility to be orphaned in a future time, which brings the problem such as double-spending. It requires an efficient retrieval mechanism for a blockchain to supervise the orphan branches in the other blockchain, which ensures that the bifurcation in the other blockchain can be detected in time.

In this paper, to overcome the above issues, we propose a supervision mechanism between blockchains, called Surveillant, which includes three protocols: listen, record, and query. In protocol: listen, we introduce new entity called dual-functional node $\mathbf{D}^{\mathcal{S}}$. $\mathbf{D}^{\mathcal{S}}$ listens for the latest state of the blockchain $\mathcal{R}$ under supervision, and maintains its real-time data. In protocol: record, $\mathbf{D}^{\mathcal{S}}$ generates new block for the supervising blockchain $\mathcal{S}$. A commitment of the real-time data of $\mathcal{R}$ will be recorded into the $\mathcal{S}$ block. By the recording process, $\mathcal{S}$ and $\mathcal{R}$ achieve blockchains synchronization, which is further illustrated in Section 6. In protocol: query, a user using an $\mathcal{S}$ client $\mathbf{U}^{\mathcal{S}}$ is able to have cross-chain verification to $\mathcal{R}$ by using the synchronization between $\mathcal{S}$ and $\mathcal{R}$. The cross-chain verification can be as nearly efficient as the verification to the local transactions in $\mathcal{S}$.

In summary, we make the following contributions:

1. We propose a supervision mechanism between blockchains. In this mechanism, new entities called dual-functional nodes are introduced to listen for the blockchain under supervision, and record the real-time commitment of it into the supervising blockchain, achieving a blockchain synchronization. Through this design, a user is able to have cross-chain verification effectively.
2. To process the large-scale blocks in the blockchain under supervision, we introduce Merkle mountain range for blocks aggregation to cut down the storage overhead. To trace the bifurcations in the blockchain under supervision, we propose the design of LOBCounter for efficient orphan branch retrieval. Moreover, the existing incentive mechanism is improved to encourage the behaviors of dual-functional nodes.
3. We analyze the security, time delay, and overhead of Surveillant in theory. Moreover, we implement a proof-of-concept prototype and evaluate the performance of Surveillant in practice.

This paper is organized as follows. In Section 2, we have a discussion about the existing schemes for blockchain interoperability. In Section 3, we review the preliminary knowledge related to our following design. We have a description about the system model, threat model, and design goals in Section 4. We have a design to the framework of Surveillant in Section 5, describe the details of blockchain synchronization in Section 6, and further analyze its security and performance of in Section 7. In Section 8, we describe the simulation experiment of Surveillant, and evaluate the result by comparing it with other schemes. Finally, we conclude in Section 9.

2. Related Work

The existing schemes for blockchain interoperability can be classified into three categories, which are notary scheme, hash-locking scheme, and chain relay scheme. In the notary scheme [10], the blockchain information is transferred by a trusted notary group, which has potential security vulnerability.

In the hash-locking scheme [11], the operations in two blockchains are sealed by the same hash lock in advance. The operations can be triggered in the same time by revealing the preimage of the hash lock, or none of the operations can be executed, which ensures the atomicity. By virtue of this property, the hash-locking scheme has been adopted in atomic cross-chain swap [12]. However, the drawback of the scheme is that the information carried by the preimage is very limited. It results that the hash-locking scheme cannot support the complex cross-chain operations which require to transfer concrete state information of a blockchain.

In the chain relay scheme such as the Ethereum project BTCRelay [13], a group of relayers, which act as the intermediaries, deliver every block header of Bitcoin to the smart contract of Ethereum. Hence, the smart contract provides the functionality comparable to a Bitcoin light client. First, based on the smart contract, the users of Ethereum are able to verify the transactions in Bitcoin, which achieves a cross-chain transaction verification. Second, because the block headers in the smart contract include the complete proof-of-work of Bitcoin, the validity of the block headers can be proved by the proof-of-work itself, which removes the dependence to trust the intermediary. For these advantages, chain relay scheme has broader application space.

In the following work, XCLAIM [14] achieves trustless cross-chain exchanges by using chain relay scheme. The cross-chain protocols in XCLAIM require a publicly verifiable audit log of user actions on both blockchains, in which the chain relay scheme plays an essential role in cross-chain verification. However, as the smart contract still has to maintain every block header of the other blockchain, the overhead of XCLAIM remains too large.

Nowadays, with the popularization of blockchain technology, the application scenario of blockchain is no longer limited to cryptocurrency, but tends to be diversified and self-organized. For example, BlocHIE [15] establishes a platform for healthcare information exchange by using two loosely-coupled blockchain. Ferrag [16] provides an overview of different application domains of blockchain technologies in internet of things, and further evaluates their security and privacy in [17]. Jiang [18] proposes a fair transaction packing algorithm for permissioned blockchain-empowered industrial IoT systems. As the different blockchain applications are built independently, one blockchain cannot guarantee the security of the other blockchain. Consequently, a corruption in a blockchain is able to influence the other blockchain if a cross-chain linking exists between them. Therefore, it is necessary for a blockchain to establish a supervision mechanism to monitor the state of the other blockchain that has a cross-chain linking with it, which prevents the exotic corruption.

3. Preliminary

3.1 Merkle Proof

In Bitcoin, the transactions in a block are aggregated by Merkle tree, and the Merkle tree root as a commitment to the transactions is recorded into the block header. In the condition when the validity of the block header has been confirmed, a given transaction in the block can be further verified by a Merkle proof (also known as SPV proof) (see **Fig. 1**) linking the transaction with the Merkle tree root in the header.

Definition 1 (Merkle Proof). *In a Merkle tree $\mathcal{T}$, a Merkle proof $\Pi_{D_i}^{\mathcal{T}}$ to data D_i is the siblings of the nodes in the Merkle tree path from $H(D_i)$ to $\mathcal{T}$ root. The length of $\Pi_{D_i}^{\mathcal{T}}$ is at most the height of $\mathcal{T}$, which equals $\lceil log_2(n) \rceil$. For the given root of $\mathcal{T}$ and the $\Pi_{D_i}^{\mathcal{T}}$ of D_i, the D_i is proved to be included in $\mathcal{T}$ when each parent node following the Merkle tree path is recursively hashed from its two children, and the final hash equals the value of $\mathcal{T}$ root.*

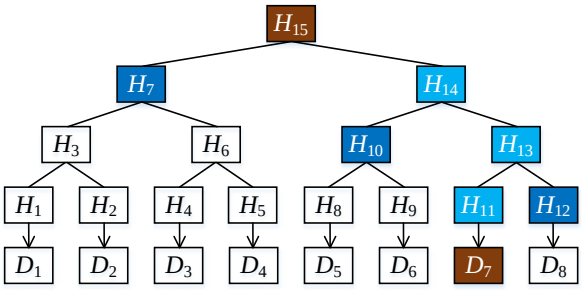


Fig. 1. Merkle proof. For the given data D_7 and the Merkle tree root hash H_{15} (marked by brown), D_7 can be proved to be included in the Merkle tree if $H_{15} = H\left(H_7 \,||\, H\left(H_{10} \,||\, H(H(D_7) \,||\, H_{12})\right)\right)$. The value H_{12}, H_{10}, and H_7 (marked by dark blue) are the Merkle proof $\Pi_{D_7}^{\mathcal{T}}$ to D_7. $\Pi_{D_7}^{\mathcal{T}}$ are the siblings of H_{11}, H_{13}, and H_{14} (marked by light blue), which are in the Merkle tree path linking D_7 to H_{15}.

3.2 Longest Chain Rule

Because of the network latency or intentional attack (see Section 3.3), different bookkeepers have disagreements to the current blockchain state. To deal with this problem, every honest bookkeeper adapts and mines on the longest chain, and the shorter block chains stop to grow, in which we say that the block chains are orphaned. The longest chain rule ensures that the bookkeepers distributed in the network tends to work on the longest chain maintained by the honest majority.

Definition 2 (Block chain). *For any given block B_j, there is only path to traverse from B_j to the genesis block B_1 in which the traversal follows the hash pointers of the blocks in the path. We define that the blocks in the path form a block chain C_j. The block chain height of C_j equals j.*

3.3 Orphan Block Phenomenon

Orphan block, sometimes referred to as stale block, are the block directly or indirectly pointing to a block in the longest chain, but not included in the longest chain. When several continuous orphan blocks hanging outside the longest chain, we call them an *orphan branch* (see **Fig. 2**). An orphan branch has two attributes, which are length and bifurcation point. A long orphan branch has the length exceeding the security threshold ϵ.

- The length of an orphan branch equals the number of the orphan blocks included in the branch.
- The bifurcation point is the block in the longest chain which the orphan branch points to.

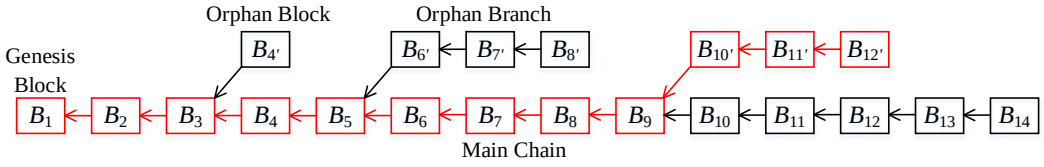


Fig. 2. Block chains and orphan branches. For any given block such as $B_{12'}$, there is only path to traverse from $B_{12'}$ to the genesis block B_1. The blocks in the path form the block chain $C_{12'}$, which is marked by red. For all of the block chains in this figure, C_{14} is known as the longest chain (For convenience, we draw only 14 blocks in the longest chain). Moreover, $B_{4'}$ is an orphan block. The orphan blocks "$\leftarrow B_{6'} \leftarrow B_{7'} \leftarrow B_{8'}$," form an orphan branch, whose length is 3, and bifurcation point is B_5. As the honest majority of bookkeepers will adapt and mine on the longest chain, the block chain $C_{12'}$ is tending to be orphaned, in which it includes the orphan branch "$\leftarrow B_{10'} \leftarrow B_{11'} \leftarrow B_{12'}$," at its tail end.

The causes of orphan block can be classified into two categories, which are network latency and intentional attack.

Network Latency. Since the network latency, the transmission of a newly generated block between nodes takes time. For this reason, there is a possibility that a group of miners having not received the newly generated block still mine on the old block chain. Consequently, a bifurcation occurs to produce two branches, and one branch will be orphaned eventually. Illustrated by Christian [19], the scale of the nodes that have received the new block has high possibility to reach majority within a certain time latency. By properly setting the block generation rate [20], the orphan blocks caused by network latency will be kept in a small scale.

Lucianna [21] further indicates that the probability of a block to be orphaned drops exponentially with its confirmation increasing. We usually assert that a block is stable when its confirmation has exceeded the threshold ϵ. In Bitcoin for example, ϵ usually defaults to 6 blocks.

Intentional Attack. Compared with the orphan blocks caused by network latency, the orphan blocks caused by intentional attacks are more harmful, such as selfish mining [22], bribery attack [23], and majority attack. In the selfish mining for example, a malicious mining pool intentionally keeps its mined blocks private, bifurcating the blockchain to create a secret branch. The secret branch is judiciously revealed when it takes chance to have more blocks than the public branch mining by the honest miners. Following the longest chain rule, the honest miners turn to mine on the revealed secret branch, leaving the public branch orphaned. Furthermore, the bifurcation may lead to a serial of problems, such as double-spending.

3.4 Full Node and Light Client

Full Node. A "full node" exists by default in a blockchain. The full node is responsible for verifying, storing and relaying the blocks and transactions on the network. Because of the trustless network environment and the nature of a blockchain, each full node needs to download and verify every single block, and therefore every single transaction in each block. Besides, the full node works full-time to keep tracking the state change of blockchain. Based on the functions of full node, organizations and individuals, such as miners, block explorers, and exchanges, run full nodes for their business.

Light Client. A light client, sometimes referred to as light node, is an end-user software that connects to full nodes to interact with the blockchain. Unlike the full node, light client does not need to work full-time or maintain the complete information of the blockchain. In a high level of security, a light client needs to download and verify every block header of the blockchain. To verify a certain transaction in a block, the light client will further download the Merkle proof linking the transaction to the header of the block. The overall verification process is achieved cryptographically, in which the light client does not need to trust the full node for every request.

4. Problem Statement

4.1 System Model

In Surveillant, one blockchain is able to have supervision to several blockchains in the same time. For convenience, in the following content, we discuss the only the supervision between two blockchains, which are the supervising blockchain $\mathcal{S}$ and the blockchain $\mathcal{R}$ under supervision. The blockchain $\mathcal{R}$ is required to be public chain, in which the blockchain data of $\mathcal{R}$ can be received by the nodes of $\mathcal{S}$. If the $\mathcal{R}$ data keeps secret like the private chain, the supervision cannot be proceeded.

First, $\mathcal{S}$ has well security and stability guarantee such as Bitcoin and Ethereum. In $\mathcal{S}$, the scale of nodes is large enough, which creates a relatively safe environment to resist an adversary to corrupt the majority of the nodes. For specific performance, the probability of a bifurcation of depth n drops exponentially with n increasing, and all transactions published by honest nodes will eventually end up at the depth more than k blocks in the blockchain, which satisfies the property of persistence and liveness [24].

However, on the other side, the blockchain $\mathcal{R}$ is not as well-equipped as $\mathcal{S}$. The blockchains like $\mathcal{R}$ are usually applied in various Altcoins or the other scenarios such as the applications in medical data access [15], and IoT [16]. In some of these situations, the scale of the participants in the blockchain is relatively limited, and the blockchain is lack of operation and maintenance. Therefore, we assume that blockchain $\mathcal{R}$ is vulnerable to intentional attack, and orphan block is much easier to be created (see Section 3.3).

The blockchain system of $\mathcal{S}$ includes two types of entities, which are dual-functional node $\mathbf{D}^{\mathcal{S}}$ and client $\mathbf{U}^{\mathcal{S}}$ (see **Fig. 3**).

Dual-functional node $\mathbf{D}^{\mathcal{S}}$. $\mathbf{D}^{\mathcal{S}}$ is the entity that generates blocks for $\mathcal{S}$, which provides the function comparable to the so-called bookkeeper [25]. Specifically, the group of $\mathbf{D}^{\mathcal{S}}$ interconnect with one another through the peer-to-peer network; $\mathbf{D}^{\mathcal{S}}$ receives and verifies the new blocks and pending transactions of $\mathcal{S}$; $\mathbf{D}^{\mathcal{S}}$ maintains the full blockchain data of $\mathcal{S}$; $\mathbf{D}^{\mathcal{S}}$ generates new blocks for $\mathcal{S}$.

In the same time, every $\mathbf{D}^{\mathcal{S}}$ listens for the full blockchain data of $\mathcal{R}$, which provides the function comparable to a listener to $\mathcal{R}$. Specifically, $\mathbf{D}^{\mathcal{S}}$ establishes connections with the nodes who do not participate in $\mathcal{S}$, but generate, transfer, and maintain the blockchain data of $\mathcal{R}$; $\mathbf{D}^{\mathcal{S}}$ receives and verifies the new blocks of $\mathcal{R}$; $\mathbf{D}^{\mathcal{S}}$ maintains the full blockchain data of $\mathcal{R}$. It is worth noted that $\mathbf{D}^{\mathcal{S}}$ belongs to the entity of $\mathcal{S}$ instead of the entity of $\mathcal{R}$, though $\mathbf{D}^{\mathcal{S}}$ receives the information from $\mathcal{R}$. In this situation, $\mathbf{D}^{\mathcal{S}}$ does not receive the pending transactions of $\mathcal{R}$, or generate new blocks for $\mathcal{R}$. Consequently, $\mathbf{D}^{\mathcal{S}}$ does not contribute to the blockchain growth of $\mathcal{R}$.

Comprehensively considering the above dual functions of $\mathbf{D}^{\mathcal{S}}$, $\mathbf{D}^{\mathcal{S}}$ is able to listen for the information of $\mathcal{R}$, and record the $\mathcal{R}$ information into $\mathcal{S}$, which achieves a commitment of $\mathcal{R}$.

From the perspective of traditional blockchain, $\mathbf{D}^{\mathcal{S}}$ can be treated as the conventional bookkeeper with the listening function added, which brings extra workload to the bookkeeper. For this problem, we propose the commitment reward for the extra workload. By this design, the improved incentive mechanism is able to encourage $\mathbf{D}^{\mathcal{S}}$ to do both the works of listening and block generating, by which the information of $\mathcal{R}$ will be committed to $\mathcal{S}$ in real time.

Clients $\mathbf{U}^{\mathcal{S}}$. $\mathbf{U}^{\mathcal{S}}$ is the end-user software of $\mathcal{S}$, in which $\mathbf{U}^{\mathcal{S}}$ is usually the light client. The main task of $\mathbf{U}^{\mathcal{S}}$ is to query the information in $\mathcal{S}$. First, $\mathbf{U}^{\mathcal{S}}$ will download every block header of $\mathcal{S}$ from $\mathbf{D}^{\mathcal{S}}$. To verify a specific transaction in $\mathcal{S}$, the $\mathbf{U}^{\mathcal{S}}$ needs to additionally download the Merkle proof of the transaction from $\mathbf{D}^{\mathcal{S}}$, which is the same with the traditional SPV. To cross-chain verify a transaction in $\mathcal{R}$, the $\mathbf{U}^{\mathcal{S}}$ needs to download a three-stage Merkle proof of the transaction. The three-stage Merkle proof includes the Merkle proves in $\mathcal{S}$, $\mathcal{R}$ and the Merkle mountain range between them (see Section 6 for more details). As $\mathbf{D}^{\mathcal{S}}$ maintains both the full blockchain data of $\mathcal{S}$ and $\mathcal{R}$, $\mathbf{D}^{\mathcal{S}}$ is able to provide the three-stage Merkle proof. In this process, $\mathbf{U}^{\mathcal{S}}$ does not need to access the entities other than $\mathbf{D}^{\mathcal{S}}$.

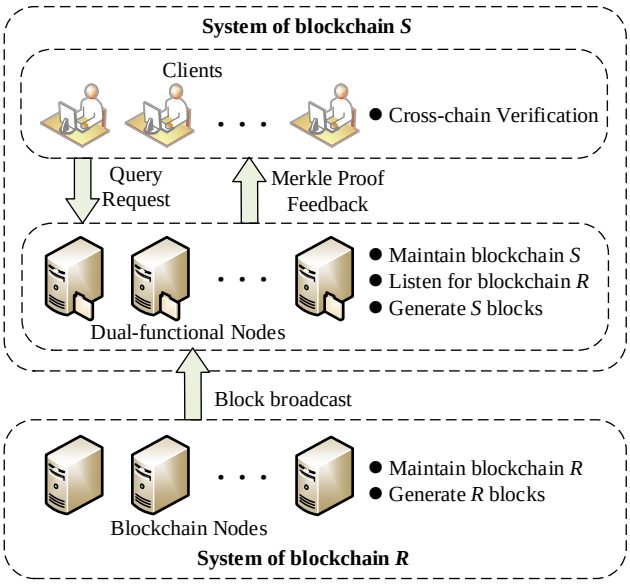


Fig. 3. Framework of Surveillant

4.2 Threat Model

We suppose an adversary is powerful enough to corrupt the blockchain $\mathcal{R}$. For example, in the blockchain based on Nakamoto consensus protocol, the adversary is able to control more than 1/3 [22] of the hash power of $\mathcal{R}$. Therefore, $\mathcal{R}$ is vulnerable to the damages such as selfish mining, bribery attack, and majority attack. Consequently, the adversary is able to bifurcate the blockchain $\mathcal{R}$ to orphan a branch and the length of the orphan branch has high possibility to exceed a threshold ϵ, which brings the problem such as double-spending. On the other side, in $\mathcal{S}$, we assume that the corrupted entities of $\mathcal{S}$ is bounded by the security threshold, such as the corrupted hash power is bounded by 1/3 in the blockchain based on Nakamoto consensus protocol. It ensures the security including the persistence and liveness of $\mathcal{S}$.

Furthermore, we assume Surveillant satisfies the following security assumptions. First, we assume the cryptographic primitives of both $\mathcal{S}$ and $\mathcal{R}$ are secure. Second, we assume that the block generation rates of $\mathcal{S}$ and $\mathcal{R}$ are upper-bounded [20], which can be achieved by periodically adjusting the difficulty targets of the two blockchains (in PoW protocol). It ensures that an adversary cannot create unlimited blocks arbitrarily. Third, we assume the blocks of $\mathcal{S}$ and $\mathcal{R}$ are broadcast in the peer-to-peer network following the information propagation model [19] [24], in which the honest majority of $\mathbf{D}^S$ will reach consensus to the newly generated block of $\mathcal{S}$ and $\mathcal{R}$ within the time delay Δ^S and Δ^R. Fourth, we assume that a user using a client $\mathbf{U}^S$ is connected to at least one honest dual-functional node $\mathbf{D}^S$, which means $\mathbf{U}^S$ is not vulnerable to eclipse attack. Defending against such attacks is beyond the scope of our paper.

4.3 Design Goals

- **Reliable information transfer**. By introducing dual-functional nodes with improved incentive mechanism, the commitment to newly generated block of $\mathcal{R}$ will be recorded into $\mathcal{S}$ in real time, and the cross-chain commitment recording resists against denial-of-service attack.

- **Efficient cross-chain verification**. The operation that a user of $\mathcal{S}$ has a cross-chain verification to $\mathcal{R}$ is as nearly efficient as verifying the local information in $\mathcal{S}$.
- **Low storage overhead**. A user of $\mathcal{S}$ has cross-chain verification without maintaining the data of $\mathcal{R}$. Subsequently, the user's extra storage overhead brought by the cross-chain requirement is negligible.

4.4 Notations

To facilitate the understanding, we summarize the main notations in this paper in **Table 1**.

Table 1. Notations

Notation	Meaning
$\mathcal{S}$	Supervising blockchain
$\mathcal{R}$	Blockchain being supervised
$\mathbf{D}^{\mathcal{S}}$	Dual-functional node of $\mathcal{S}$
$\mathbf{U}^{\mathcal{S}}$	Client of $\mathcal{S}$
$B_i^{\mathcal{S}}$	Block in $\mathcal{S}$, in which i equals the block's height
$B_j^{\mathcal{R}}$	Block in $\mathcal{R}$, in which j equals the block's height
$C_j^{\mathcal{R}}$	Block chain in $\mathcal{R}$ formed by the blocks from $B_j^{\mathcal{R}}$ to $B_1^{\mathcal{R}}$
$\pi_{ij}^{\mathcal{R}}$	Commitment to $C_j^{\mathcal{R}}$, in which $\pi_{ij}^{\mathcal{R}}$ is recorded into $B_i^{\mathcal{S}}$
$T^{\mathcal{S}}$	Transaction in $\mathcal{S}$
$T^{\mathcal{R}}$	Transaction in $\mathcal{R}$

5. Surveillant Framework

5.1 Overview

In this section, we present Surveillant: an efficient supervision mechanism between blockchains. For Surveillant in general, the dual-functional node $\mathbf{D}^{\mathcal{S}}$ listens for the blockchain information of $\mathcal{R}$, generates commitment to the information, and records the commitment into $\mathcal{S}$. Subsequently, the clients $\mathbf{U}^{\mathcal{S}}$ is able to have efficient and low-cost cross-chain verification to the information through the commitment recorded in $\mathcal{S}$.

The basic scheme of Surveillant includes three protocols: listen, record, and query. (see **Fig. 4**)

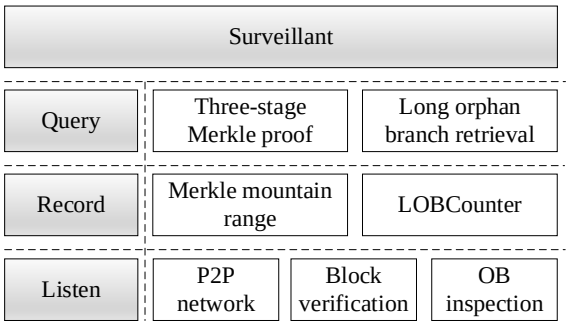


Fig. 4. Architecture of Surveillant

5.2 Protocol: Listen

$\mathbf{D}^{\mathcal{S}}$ receives and maintains the full blockchain data of $\mathcal{R}$. Subsequently, $\mathbf{D}^{\mathcal{S}}$ has verification to the blocks received and inspect any orphan branches occurred in $\mathcal{R}$.

Block verification. $\mathbf{D}^{\mathcal{S}}$ has verification to the correctness of each new $B_j^{\mathcal{R}}$ received. First, the block header of $B_j^{\mathcal{R}}$ is verified based on the consensus protocol which blockchain $\mathcal{R}$ is using. For PoW consensus protocol, $\mathbf{D}^{\mathcal{S}}$ is required to know the difficulty policy of $\mathcal{R}$, and verify the correctness of proof-of-work in $B_j^{\mathcal{R}}$. For PoS consensus protocol, such as Ouroboros [1], $\mathbf{D}^{\mathcal{S}}$ is required to know the leader selecting epochs, and verify the correctness of leader signature in $B_j^{\mathcal{R}}$. Second, the transactions in $B_j^{\mathcal{R}}$ is verified, in which the inclusion of a transaction in $B_j^{\mathcal{R}}$ is required to be proved by a Merkle proof linking the transaction to the root in $B_j^{\mathcal{R}}$ header.

Orphan branch inspection. Discussed in Section 3.3, orphan blocks are the blocks not included in the longest chain, and several continuous orphan blocks form an orphan branch. If the length of an orphan branch has exceeded a threshold ϵ, the branch is judged as a long orphan branch, which brings the problem such as double-spending. In this situation, $\mathbf{D}^{\mathcal{S}}$ will verify the long orphan branch generated in $\mathcal{R}$, and record its information into $\mathcal{S}$.

5.3 Protocol: Record

In this process, a commitment of $\mathcal{R}$ will be recorded into each $\mathcal{S}$ block. The recording process includes two steps, which are commitment recording and commitment verification.

Commitment recording. First, at the moment when an $\mathcal{S}$ block $B_i^{\mathcal{S}}$ is being generated by a $\mathbf{D}^{\mathcal{S}}$, the $\mathcal{R}$ block chain $C_j^{\mathcal{R}}$ with the largest length among the current $\mathcal{R}$ chains maintained by the $\mathbf{D}^{\mathcal{S}}$ itself is committed by using the data structure of Merkle mountain range [26]. Then the commitment $\pi_{ij}^{\mathcal{R}}$ is recorded into the Merkle tree of $B_i^{\mathcal{S}}$.

Second, for the $\mathcal{R}$ chain $C_{j'}^{\mathcal{R}}$ committed by $\pi_{(i-1)j'}^{\mathcal{R}}$, $(j' < j)$ in $B_{i-1}^{\mathcal{S}}$, if the blocks at the tail end of $C_{j'}^{\mathcal{R}}$ are not included in $C_j^{\mathcal{R}}$, these blocks are orphaned to form an orphan branch. In this situation, if the length of the orphan branch exceeds the threshold ϵ, we have the assignments in which $\text{LOBCounter}_i \leftarrow \text{LOBCounter}_{i-1} + 1$, and $\text{BCHeight}_i \leftarrow j$.

LOBCounter_i is a field added in the block header of $B_i^{\mathcal{S}}$. It counts the long orphan branches (LOB) that occurred in the history of $\mathcal{R}$. BCHeight_i is also a field in $B_i^{\mathcal{S}}$ header. It records the height of each committed $\mathcal{R}$ block chain. The two fields enable an efficient orphan branch retrieval, which is discussed in the following content.

Commitment verification. Next, the generated $B_i^{\mathcal{S}}$ is broadcast to the other $\mathbf{D}^{\mathcal{S}}$ to reach consensus. First, a $\mathbf{D}^{\mathcal{S}}$ having received $B_i^{\mathcal{S}}$ will verify that the $\pi_{ij}^{\mathcal{R}}$ in $B_i^{\mathcal{S}}$ must commit to one of the $\mathcal{R}$ block chains maintained by the $\mathbf{D}^{\mathcal{S}}$ itself, in which the $\mathcal{R}$ block chains are received by the $\mathbf{D}^{\mathcal{S}}$ during the listening process. Second, BCHeight_i must equal the height of $C_j^{\mathcal{R}}$, and LOBCounter_i must be accumulated by one if a long orphan branch is created.

Subsequently, the $\mathcal{S}$ block having correct format and block content will be included into blockchain $\mathcal{S}$. As each $\mathcal{S}$ block records an $\mathcal{R}$ commitment, $\mathcal{S}$ and $\mathcal{R}$ are synchronized, which enables efficient cross-chain verification.

5.4 Protocol: Query

By using the synchronization between $\mathcal{S}$ and $\mathcal{R}$, a user using an $\mathcal{S}$ client $\mathbf{U}^S$ is able to have cross-chain verification to $\mathcal{R}$.

First, based on a recent stable $\mathcal{S}$ block B^S recording the commitment π^R, the user cross-chain verifies a transaction T^R in an $\mathcal{R}$ block B^R by downloading a three-stage Merkle proof, which is the connection of the Merkle proof of π^R in the Merkle tree of B^S, the Merkle proof of B^R in the Merkle mountain range of π^R, and the Merkle proof of T^R in the Merkle tree of B^R. By this design, the transaction in $\mathcal{R}$ can be cross-chain verified efficiently without maintaining the data in $\mathcal{R}$.

Second, by checking where LOBCounter is accumulated in $\mathcal{S}$, the user is able to retrieve any long orphan branches occurred in the history of $\mathcal{R}$. Moreover, BCHeight helps locating the position of the branches. This retrieval can be achieved by only downloading the block headers of $\mathcal{S}$.

6. Blockchains Synchronization

6.1 Motivation

Discussed in Section 3.1, blockchain is a distributed digital ledger of cryptographically signed transactions that are grouped into blocks. Each block header records a hash pointer pointing to the header of the last block, and the transactions is grouped by the Merkle tree in a block, in which the hash pointer and Merkle tree constitute the cryptographical links among each part of the blockchain data structure. Therefore, a user using a blockchain light client is able to have cryptography-based verification to any blocks and transactions in the blockchain. The advantage of cryptography-based verification is that it is efficient in operation, and it does not need to trust the intermediaries.

However, as different blockchains are mutually independent in data structure, there are no cryptographical links between them. This limitation results that we cannot use the same cryptography-based verification method to achieve cross-chain verification, in which the cross-chain verification refers that a user using a client of blockchain $\mathcal{S}$ wants to verify the information in blockchain $\mathcal{R}$. For the other methods, some existing schemes have cross-chain verification with the help of trusted third parties, or each user using a client of $\mathcal{S}$ additionally maintains the data of $\mathcal{R}$ to verify both of the blockchains in the same time. However, the existing methods are insecure or costly.

In respect of the issues above, if we establish the cryptographical links between the two blockchains by embedding a real-time blockchain commitment of $\mathcal{R}$ into every block of $\mathcal{S}$, the application of cryptography-based verification method to cross-chain verification will become feasible. This conception will have two advantages. First, a user using a client of $\mathcal{S}$ will be able to have cross-chain verification to a transaction in $\mathcal{R}$ through the blockchain commitment π^R embedded in $\mathcal{S}$, which is as nearly efficient as the verification to a transaction in $\mathcal{S}$. Second, the user does not need to additionally maintain the data of $\mathcal{R}$, avoiding the high usage costs.

6.2 Overview

Starting from the idea mentioned above, we achieve synchronization between $\mathcal{S}$ and $\mathcal{R}$ by blockchain commitment, enabling efficient cross-chain verification.

First, the $\mathcal{R}$ block chain C_j^R is committed by using Merkle mountain range (MMR), which is a variant of Merkle tree. In C_j^R, the blocks from B_1^R to B_j^R is grouped by MMR, and the root

of the MMR is the commitment π_{ij}^{R} to C_j^R. By virtue of MMR, the size of π_{ij}^{R} can be as small as the size of a common transaction.

At the moment when a new $\mathcal{S}$ block B_i^S is generated by a $\mathbf{D}^S$, the π_{ij}^{R} is recorded into B_i^S. Subsequently, with i and j increasing, the successively generated B_i^S record the $\pi_i^R j$ committing to C_j^R with increasing length, describing the state update of $\mathcal{R}$. By recording the blockchain commitment, $\mathcal{S}$ and $\mathcal{R}$ are synchronized.

To encourage the blockchain commitment, we design an incentive mechanism. In this design, the $\mathbf{D}^S$ committing the C_j^R with larger length will earn more commitment reward. In order to maximize revenue, the honest majority will be initiative to listen to the most real-time information of $\mathcal{R}$, and commit the information to $\mathcal{S}$, achieving active blockchain commitment.

Through the synchronization between $\mathcal{S}$ and $\mathcal{R}$, a user using an $\mathcal{S}$ client $\mathbf{U}^S$ is able to have cross-chain verification to $\mathcal{R}$. The cross-chain verification does not need to rely the trusted third parties, and the cost to have a cross-chain verification has been nearly close to the cost to have a verification to the local information in $\mathcal{S}$. Therefore, the user is able to have on-demand cross-chain verification, breaking through the limitation of trust and cost.

6.3 Blockchain Commitment

Because the data in a blockchain occupies a lot of storage space, for example the Bitcoin blockchain takes up about 340 GB. It is completely impractical to directly commit the untreated $\mathcal{R}$ data into $\mathcal{S}$ block. A preprocessing method is to take the hash of every $\mathcal{R}$ block. However, the size of the block hashes is still too large to commit. For example, all block hashes of Bitcoin blockchain takes up about 21 MB.

Merkle mountain range. To solve the above problem, we first introduce the data structure of MMR. Considering both the definition of MMR, each $\mathcal{T}_k$ in MMR is actually a Merkle tree. Subsequently, the data D_j can be proved to be included in MMR by a Merkle proof (see Definition 1), and the complexity of the Merkle proof is $O(\log n)$.

Definition 3 (Merkle Mountain Range). *Merkle mountain range is formed by a sequence of perfect binary trees $\mathcal{T}_k$ with strictly decreasing heights h_k. In a $\mathcal{T}_k$, each leaf of $\mathcal{T}_k$ includes a data hash $H(D_j)$, and each non-leaf node includes a hash $H(LeftChild \,||\, RightChild)$. The number of all leaves in MMR equals $\sum 2^{h_k}$. The hash of the $\mathcal{T}_k$ roots is the root of MMR. The hash function used in MMR is collision resistant.*

Compared with Merkle tree, MMR has the advantage of extendibility. When new data hashes are added as new leaves are appended in MMR, we do not have to recalculate all the hashes in MMR, but generate the new $\mathcal{T}_{k''}$ roots ($k < k''$) based on the old $\mathcal{T}_k$ roots along with the new leaves.

Commitment to $\mathcal{R}$. To make a smaller $\mathcal{R}$ commitment to be recorded into $\mathcal{S}$ block, we further aggregate the hashes of the $\mathcal{R}$ blocks by using MMR (see **Fig. 5**).

For a chosen $\mathcal{R}$ block chain C_j^R with the largest length, the hashes of the blocks in C_j^R from B_1^R to B_j^R are sequentially included into the leaves of a MMR, and the MMR root is included in the commitment π_{ij}^{R}, in which π_{ij}^{R} has the size comparable to the size of a common transaction. Now π_{ij}^{R} is small enough to be recorded into B_i^S.

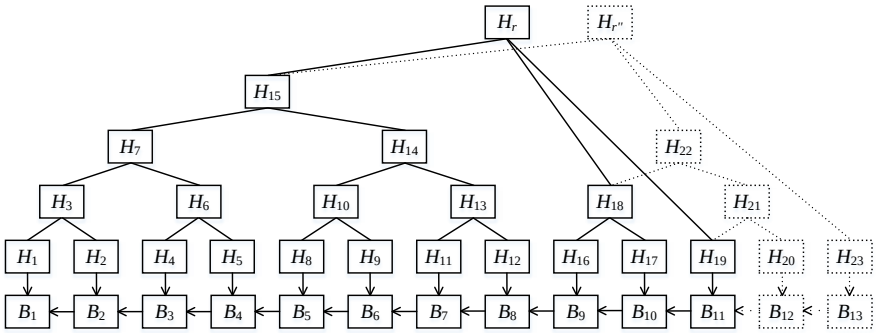


Fig. 5. Merkle Mountain Range. The blocks from B_1 to B_{11} in a block chain are aggregated by the MMR (marked by full line). For convenience, we draw only 11 blocks here. The binary trees with the roots H_{15}, H_{18}, and H_{19} are three Merkle trees, and the hash $H_r = H(H_{15} \,||\, H_{18} \,||\, H_{19})$ is the root of the MMR. When new blocks B_{12} and B_{13} (marked by dotted line) are added into the block chain, the new hashes H_{20}, H_{21}, H_{22}, H_{23}, and $H_{r''}$ are generated based on H_{15}, H_{18}, and H_{19}, and $H_{r''}$ is the root of the new MMR.

6.4 Synchronization via Blockchain Commitment

Discussed in Section 5, at the moment when a new block B_i^S is assembled by a dual-functional node $\mathbf{D}^S$, the π_{ij}^R committed by the $\mathbf{D}^S$ is recorded into B_i^S. After finding a nonce to make the B_i^S satisfy the difficulty, the $\mathbf{D}^S$ broadcasts the B_i^S to other $\mathbf{D}^S$ to reach consensus.

Commitment verification. In the process of consensus, an honest $\mathbf{D}^S$ having received the B_i^S is required to have verification to the π_{ij}^R in it. A valid π_{ij}^R satisfies the following two conditions:

- Congruity: π_{ij}^R commits to an $\mathcal{R}$ block chain maintained by the honest $\mathbf{D}^S$ itself.
- Monotonicity: the length of C_j^R is greater than or equal to the length of $C_{j'}^R$ committed by the $\pi_{(i-1)j'}^R$ in B_{i-1}^S.

For the congruity, if π_{ij}^R does not commit to any block chains maintained by the honest $\mathbf{D}^S$, the $\mathbf{D}^S$ affirms that the π_{ij}^R is not true, and the π_{ij}^R will not generated new $\mathcal{S}$ block following B_i^S. It is worth noted if a block chain C_j^R committed by π_{ij}^R is orphaned in a future time, the π_{ij}^R is still true. Because the π_{ij}^R reflects the state of $\mathcal{R}$ in a certain time period. This commitment enables us to locate the long orphan branches occurred in $\mathcal{R}$.

For the monotonicity, as the blockchain $\mathcal{R}$ increases with new block extended, the $\mathbf{D}^S$ should follow the increment to commit the $\mathcal{R}$ block chain with increasing length. Otherwise, the behavior committing the out-of-date block chains with decreasing length violate our intention to deliver the real-time information from $\mathcal{R}$ to $\mathcal{S}$.

Synchronization between $\mathcal{S}$ and $\mathcal{R}$. The B_i^S satisfying the condition of congruity and monotonicity will reach consensus among the honest majority, and be included into $\mathcal{S}$. As a result, the blockchain $\mathcal{S}$ and $\mathcal{R}$ are synchronized to form an integrated three-layer data structure (see **Fig. 6**): the first layer is $\mathcal{S}$, whose block records π_{ij}^R; the second layer is π_{ij}^R, which commits to $\mathcal{R}$ by using MMR; the third layer is $\mathcal{R}$, which records its own transactions. The synchronization between $\mathcal{S}$ and $\mathcal{R}$ enables the cross-chain transaction through a three-stage Merkle proof, which is discussed in Section 6.6.

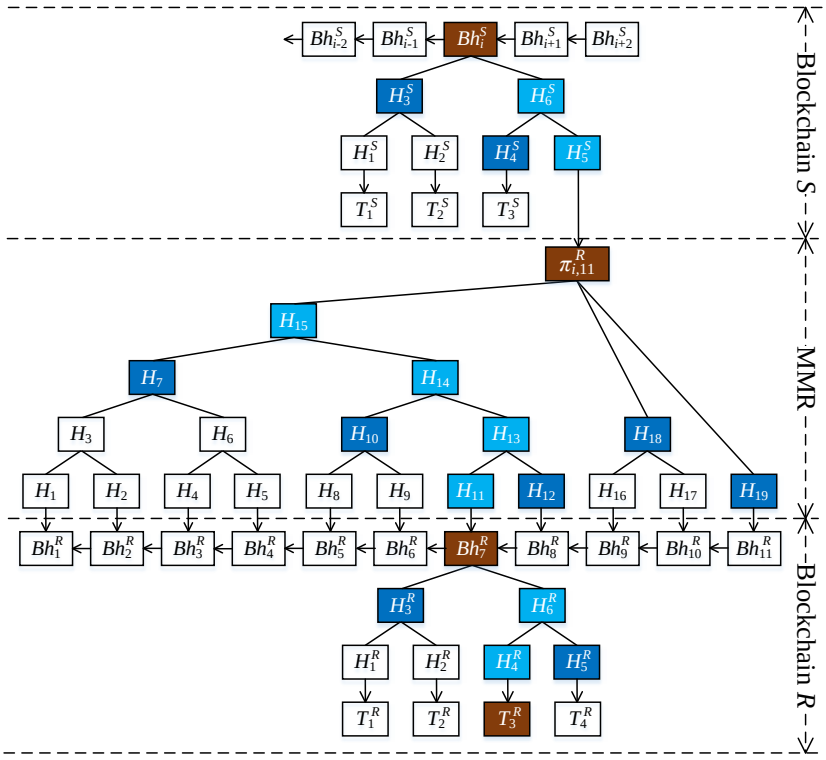


Fig. 6. Synchronization between $\mathcal{S}$ and $\mathcal{R}$. *$\mathcal{S}$ and $\mathcal{R}$ are synchronized to form a three-layer data structure, which includes the blockchain $\mathcal{S}$, MMR, and blockchain $\mathcal{R}$. For convenience, we draw only the block headers of $\mathcal{S}$ and $\mathcal{R}$ with necessary block contents. Based on a stable $\mathcal{S}$ block such as B_i^S with the block header Bh_i^S, a user cross-chain verifies a given transaction such as T_3^R by the three-stage Merkle proof (marked by dark blue). First, the validity of $\pi_{i,11}^R$ is proved, when the Merkle tree root in Bh_i^S equals $H\left(H_3^S \,||\, H\left(H_4^S \,||\, H\left(\pi_{i,11}^R\right)\right)\right)$. Second, the validity of Bh_7^R is proved, when the MMR root in $\pi_{i,11}^R$ equals $H\left(H\left(H_7 \,||\, H\left(H_{10} \,||\, H(H(Bh_7^R) \,||\, H_{12})\right)\right) \,||\, H_{18} \,||\, H_{19}\right)$. Third, the validity of T_3^R is proved, when the Merkle tree root in Bh_7^R equals $H\left(H_3^R \,||\, H(H(T_3^R) \,||\, H_5^R)\right)$.*

Treatment to orphan branch. Because of the network latency or intentional attack (discussed in Section 3.3), the $\mathcal{R}$ block chain C_j^R committed by π_{ij}^R has the possibility to be orphaned in a future time. Specifically, the honest miners of $\mathcal{R}$ adapt and mine on another $\mathcal{R}$ block chain with the largest length, and C_j^R stops to increase leaving an orphan branch at the tail end of C_j^R.

In this situation, the honest majority of $\mathbf{D}^S$ will detect this orphaning event, and preserve C_j^R along with its orphan branch. The π_{ij}^R committing to C_j^R still satisfies the condition of congruity, although C_j^R has been orphaned. The commitment to orphan branch is not trivial, as it reflects the state of $\mathcal{R}$ in different time period.

More importantly, when the length of an orphan branch has exceeded a threshold ϵ, the probability that the long orphan branch is caused by network latency drops to nearly zero, which means that the long orphan branch is very likely to be caused by intentional attack. Hence, the commitment to long orphan branch reflects the malicious behaviors in $\mathcal{R}$. Moreover, the LOBCounter_i counting the long orphan branches will provide marks for users to locate the

drastic structure changes caused by intentional attacks.

6.5 Incentive for Blockchain Commitment

The work to commit the $\mathcal{R}$ block chain brings additional workload without retribution. Although the condition of monotonicity has regulated that a $\mathbf{D}^S$ is not allowed to commit the out-of-date $\mathcal{R}$ block chains with decreasing length, the commitment to the up-to-date $\mathcal{R}$ block chains with larger length is not encouraged by the regulation. It results that the $\mathbf{D}^S$ will be passive to do the commitment, and the latest information in $\mathcal{R}$ cannot be committed in real time.

To achieve active blockchain commitment, we propose the "commitment reward" to incentive the commitment, which is inspired by the incentive mechanism in Bitcoin. In Bitcoin, there are two types of rewards, which are the block reward and transaction fee. For the block reward, a bookkeeper who generates a new block will receive 6.25 Bitcoin, in which 6.25 Bitcoins is created out of nothing by a coin-creation transaction in the new block. For the transaction fee, a user who publishes a transaction will attach a transaction fee on it, rewarding a bookkeeper to timely record the transaction into the new block.

Commitment reward. As the third type of reward, the commitment reward is created out of nothing by a coin-creation transaction T_c^S, which has the same creation process as the block reward in Bitcoin. The T_c^S is recorded into B_i^S along with the other transactions in B_i^S, and the reward is sent to a recipient address chosen by the $\mathbf{D}^S$ who generates the B_i^S.

The value of commitment reward monotonically increases with the proof-of-work in the fresh blocks committed by π_{ij}^R. To get more commitment reward, a $\mathbf{D}^S$ should actively obtain the new fresh blocks mined in the longest chain of $\mathcal{R}$, and timely commit the fresh blocks before other $\mathbf{D}^S$ do. Subsequently, $\mathbf{D}^S$ is incentivized to commit the real-time information of $\mathcal{R}$ to $\mathcal{S}$.

Definition 4 (Fresh block). *Fresh blocks are the newly generated $\mathcal{R}$ blocks which are committed by π_{ij}^R, but were not committed by any $\pi_{i'j'}^R$ before π_{ij}^R ($i' < i, j' \leq j$).*

Backoff against network latency. However, the behavior that $\mathbf{D}^S$ try to commit the latest $\mathcal{R}$ block brings additional problem in the consensus process of B_i^S. Because of the network latency, the transmission of the $\mathcal{R}$ block takes time, in which we assume that the honest majority of $\mathbf{D}^S$ will reach consensus to the newly generated block of $\mathcal{S}$ and $\mathcal{R}$ within the maximum time delay Δ^S and Δ^R. There is a phenomenon that the B_i^S with π_{ij}^R committing to the B_j^R has been spread throughout the network, but B_j^R has not been received by the honest majority of $\mathbf{D}^S$. This phenomenon is more common when the $\mathcal{S}$ block propagation speed is higher than the propagation speed of $\mathcal{R}$ block, or in another word $\Delta^S < \Delta^R$.

Consequently, there can be a time interspace in which the π_{ij}^R in B_i^S is not congruent with any $\mathcal{R}$ block chains maintained by the honest majority, as most of the honest $\mathbf{D}^S$ have not received B_j^R. The honest $\mathbf{D}^S$ will affirm that π_{ij}^R does not satisfy the condition of congruity, and chooses not to generate new $\mathcal{S}$ block following B_i^S. This affirmation keeps until B_j^R has reached consensus. In this time interspace, B_i^S takes the risk to be orphaned.

To lower the risk, A $\mathbf{D}^S$ who generates the $\mathcal{S}$ block will have backoff against network latency. The $\mathbf{D}^S$ does not directly commit to the latest $\mathcal{R}$ block after first receiving it. Instead, the $\mathbf{D}^S$ waits for a certain time, and let the $\mathcal{R}$ block reach consensus among the majority of $\mathbf{D}^S$.

Then the $\mathbf{D}^S$ commits the latest $\mathcal{R}$ block.

6.6 Cross-chain Verification via Synchronization

As the blockchains $\mathcal{S}$ and $\mathcal{R}$ have been synchronized via the blockchain commitment, a user using an $\mathcal{S}$ client $\mathbf{U}^S$ is able to have cross-chain verification via the synchronization. The cross-chain verification includes two aspects, which are cross-chain transaction verification and long orphan branch supervision.

Cross-chain transaction verification. First, the user chooses a recent stable B_i^S with large enough confirmation. A valid B_i^S header satisfies that every $B_{i'}^S$ prior to B_i^S $(i' < i)$ is correctly pointed by the hash pointer in $B_{i'+1}^S$, which ensures no manipulation occurs. Based on the B_i^S header, the user further verifies a transaction T^R in $\mathcal{R}$ through a three-stage Merkle proof across $\mathcal{S}$ and $\mathcal{R}$ (see **Fig. 6**).

- The first stage Merkle proof is the hash path linking the B_i^S header to the π_{ij}^R in B_i^S. The Merkle proof in this stage is situated in the Merkle tree in B_i^S, as the π_{ij}^R is recorded along with other transactions in B_i^S.
- The second stage Merkle proof is the hash path linking the π_{ij}^R to the header of the $\mathcal{R}$ block B_j^R. The Merkle proof in this stage is situated in the MMR.
- The third stage Merkle proof is the hash path linking the B_j^R header to the target transaction T^R in B_j^R. The Merkle proof in this stage is situated in the Merkle tree in B_j^R.

The T^R is verified by doing hash calculations backward along the three-stage Merkle proof, which is from T^R to B_j^R header, to π_i^R, and to B_i^S header. The T^R is valid when the result of the hash calculations matches the Merkle root in B_i^S header.

Long orphan branch supervision. In the second aspect, the user using the $\mathbf{U}^S$ has supervision to the long orphan branches in $\mathcal{R}$ by checking the LOBCounter_i in B_i^S. An accumulation of LOBCounter_i indicates that a long orphan branch whose length has exceeded the threshold ϵ is created in $\mathcal{R}$, which reflects that an intentional attack occurs in $\mathcal{R}$ with large possibility. Subsequently, based on all the B_i^S with the LOBCounter_i accumulated, a user is able to locate the long orphan branches occurred in $\mathcal{R}$ to trace the malicious behaviors in the history of $\mathcal{R}$.

Simplified re-verification. A combination of the cross-chain transaction verification and long orphan branch supervision can further simplify the re-verification to the transactions in $\mathcal{R}$.

There is a common scenario that a business proceeded in $\mathcal{S}$ is related to a T^R in $\mathcal{R}$, and the prerequisite of every operation in the business is that the T^R must stably exist in $\mathcal{R}$. However, as each block in $\mathcal{R}$ has the possibility to be orphaned in a future time, the T^R recorded in an $\mathcal{R}$ block may be removed later. Therefore, to guarantee the validity of every operation in the business, the business participants have to re-verify the T^R every time in each operation, ensuring that the T^R exists in $\mathcal{R}$.

We assume that a business participant first verifies the T^R based on the header of B_i^S, and the current latest stable block of $\mathcal{S}$ is $B_{i''}^S$ $(i < i'')$. By virtue of the long orphan branch supervision, the multiple re-verifications to the T^R can be omitted if T^R satisfies the following two conditions.

- First, the value $\text{BCHeight}_{i''-1} - \text{BCHeight}_i$ has exceeded threshold ϵ.
- Second, the $\text{LOBCounter}_{i''}$ in $B_{i''}^S$ does not accumulated compared with the LOBCounter_i in B_i^S.

The two condition means that the confirmation of B_j^R recording the T^R is at least larger than ϵ, and no orphan branch with large enough length occurred to remove B_j^R. Therefore, the T^R has been showed to exist in $\mathcal{R}$, and the re-verification to the T^R is not necessary.

7. Analysis

7.1 Security Analysis

Illegal blockchain commitment. While generating the commitment to $\mathcal{R}$, a dishonest $\mathbf{D}^S$ may tamper the block hash of $\mathcal{R}$, create fabricated hash pretending to be the hash of the new $\mathcal{R}$ block, or just arbitrarily commit a random value to create a false commitment. It results that none of the $\mathcal{R}$ block chains correspond with the commitment, which violates the condition of congruity. Second, a dishonest $\mathbf{D}^S$ may record an out-of-date commitment π_{ij}^R committing to an $\mathcal{R}$ block chain with smaller length compared with the $\mathcal{R}$ block chain committed by $\pi_{(i-1)j''}^R$ ($j < j''$), which violates the condition of monotonicity. We assume that the honest majority of $\mathbf{D}^S$ will reach consensus to an $\mathcal{R}$ block within the time delay Δ^R. Subsequently, the honest majority are able to verify the false and out-of-date commitment based on the received information of $\mathcal{R}$, and the $\mathcal{S}$ block recording the illegal commitment will not be included into $\mathcal{R}$.

Counterfeit three-stage Merkle proof. First, we review the security of Merkle proof. For a given Merkle tree $\mathcal{T}$, $\mathcal{T}$ is constructed by collision-resistant hash function. The root of $\mathcal{T}$ is $r^{\mathcal{T}}$, every node in $\mathcal{T}$ has a path to $r^{\mathcal{T}}$, and every Merkle proof must end with $r^{\mathcal{T}}$. Assume a PPT adversary is able to forge a Merkle proof $\Pi_D^{\mathcal{T}}$ for a D not in $\mathcal{T}$. It is required that the adversary establishes a new path linking D with a node $H(x \,||\, y)$ in $\mathcal{T}$ which has an initial path to $r^{\mathcal{T}}$, and finds an x' (or y') to satisfy $H(x' \,||\, y) = H(x \,||\, y)$ and $x' \,||\, y \neq x \,||\, y$ (or $H(x \,||\, y') = H(x \,||\, y)$ and $x \,||\, y' \neq x \,||\, y$), in which a collision occurs.

Furthermore, the three-stage Merkle proof is constituted by three Merkle proofs with coherent path in three Merkle trees. The path links the transaction T^R to the block header of $\mathcal{S}$, traversing across $\mathcal{R}$ block, MMR, and $\mathcal{S}$ block. Given a stable $\mathcal{S}$ block header which has been confirmed valid, an adversary forging a three-stage Merkle proof for a transaction T not in $\mathcal{R}$ is required to establishes a new path linking T to a node in any one of the three Merkle trees. However, it is hard to find a collision in any node.

Denial-of-service attack. An adversary may try to control the $\mathbf{D}^S$ to suspend or delay the commitment to $\mathcal{R}$. However, the decentralized design of Surveillant makes the denial-of-service attack difficult. To permanently exclude the $\mathcal{S}$ block recording the commitment, the adversary has to have a majority attack. It creates the adversary's $\mathcal{S}$ chain without recording the $\mathcal{R}$ commitment, and the chain has overwhelming advantage in length to exclude the other block chains generated by the honest majority. To delay the commitment to real-time information of $\mathcal{R}$, the adversary has to have bifurcation in $\mathcal{S}$ to orphan the $\mathcal{S}$ blocks with large enough confirmation. However, the condition for the above attacks is that the corrupted hash power in $\mathbf{D}^S$ should have exceeded the security threshold, which violates our initial assumption.

7.2 Time Consumption of Blockchain Commitment

The time consumption is calculated from the generation of B_j^R to the moment when π_{ij}^R committing to C_j^R is recorded into the newly generated B_i^S. First, we assume that B_j^R is broadcast in network following the block propagation model $P(t)$ [19], in which the proportion of the honest majority that have received the B_j^R increases with time t following the model $P(t)$. In the condition without backoff, the $\mathbf{D}^S$ having received B_j^R begin to generate B_i^S immediately. The total hash rate of all $\mathbf{D}^S$ is h, $\bar{H}$ is the average number of hashes required to generate an $\mathcal{S}$ block, and Δt is the time it takes. Their relationship satisfies the following equation. Considering the backoff time Δt_b taken by $\mathbf{D}^S$, the average time consumption of blockchain commitment equals $\Delta t + \Delta t_b$.

$$h \int_0^{\Delta t} P\,(t), dt = \bar{H}$$

7.3 Storage and Bandwidth Overhead

In addition to the function to receive, maintain and generate the blocks of $\mathcal{R}$, $\mathbf{D}^S$ is required to receive and maintain the full blockchain data of $\mathcal{R}$, which brings additional overhead to storage and bandwidth. However, compared with the mining process in $\mathcal{S}$ that consumes most of the resource of $\mathbf{D}^S$, the additional overhead appears to be insignificant.

On the other hand, a user using a light client $\mathbf{U}^S$ downloads the three-stage Merkle proof to have cross-chain verification to a transaction in $\mathcal{R}$. Because the complexity of Merkle proof increases logarithmically, and the user does not need to download or maintain the data on $\mathcal{R}$. The cross-chain verification to a transaction in $\mathcal{R}$ can be as nearly efficient as the verification to a local transaction in $\mathcal{S}$.

8. Implementation and Evaluation

8.1 Implementation

We implement a simulation experiment for Surveillant. The main processes of Surveillant are written in Python (version 3.8.3) Javascript(node.js), and HTML. The blockchain $\mathcal{R}$ is deployed on Intel Xeon E5-2680 v4 CPU @2.4GHz, 32GB DDR, and Ubuntu 18.04 64bit operating system. 100,000 dual-functional nodes of $\mathcal{S}$ are deployed on Intel Xeon E5-2680 v4 CPU @2.4GHz, 32GB DDR, and Ubuntu 18.10 64bit operating system. The clients of $\mathcal{S}$ are deployed on Intel Core i7-7500U @2.70GHz, and Windows 10.0.19042 64bit operation system.

8.2 Result Analysis

First, for blockchain $\mathcal{R}$ and $\mathcal{S}$, we investigate how long it takes for an $\mathcal{R}$ block to be committed to $\mathcal{S}$. On the side of $\mathcal{R}$, the propagation of $\mathcal{R}$ block is improvable, as the application of delegations or gossip protocol will accelerate the speed. The block propagation time is defined as the time required for a newly generated $\mathcal{R}$ block to reach consensus among the honest majority. On the side of $\mathcal{S}$, the block generation rate is adjustable following different difficulty policies or consensus epoch setting. The time it takes for committing is calculated from the moment when an $\mathcal{R}$ block is generated to the moment when an $\mathcal{S}$ block which records the commitment committing to the $\mathcal{R}$ block is generated. Measurements are given as the

average over 1000 test runs, and the results are presented in **Fig. 7**. From the results, we can find that the time consumption of blockchain commitment is mainly affected by block generation rate (BGenRate). The block propagation time has slight influence on the time consumption, as the block propagation time is relatively smaller compared with the block generating time.

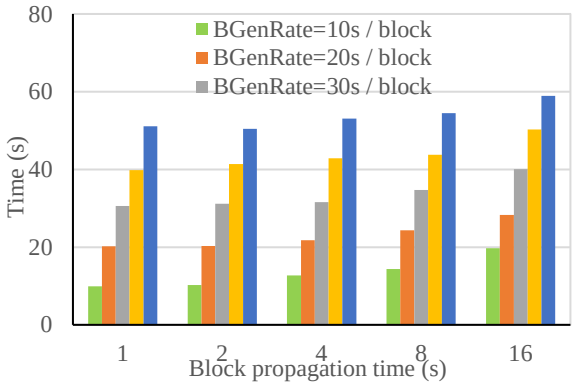


Fig. 7. Time consumption of blockchain commitment

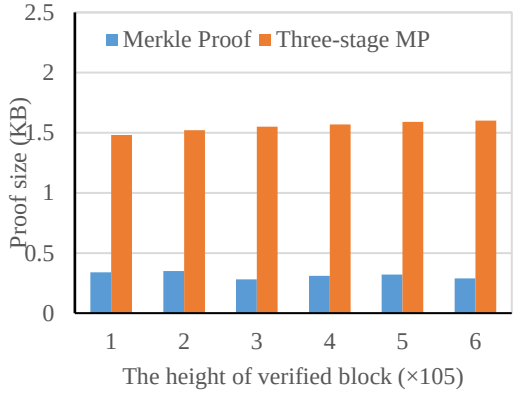


Fig. 8. Size of proves

Second, we have a comparison about the user's storage overhead in Chain relay, XCLAIM, and Surveillant. The characteristics of blockchain $\mathcal{S}$ and $\mathcal{R}$ are configured referring to the specifications of Ethereum [27] and Bitcoin [28]. The trusted checkpoint is built into the code of the $\mathcal{S}$ client as Ethereum design, by which a user using a light client of $\mathcal{S}$ only need to download the latest headers of $\mathcal{S}$. We measured the size of total data stored by the user, as shown in **Table 2**. Compared with Chain relay and XCLAIM that maintain both the block headers of $\mathcal{S}$ and $\mathcal{R}$, the client of Surveillant is smaller than the clients of the other two schemes, as Surveillant does not need to store the block headers of $\mathcal{R}$.

Table 2. Storage overhead

	Chain relay	XCLAIM	Surveillant
Client size (MB)	165	179	104

Third, we compare the bandwidth overhead of Chain relay, XCLAIM, and Surveillant. It is measured by the the average data size which a user needs to download for a cross-chain transaction verification. In $\mathcal{S}$ and $\mathcal{R}$, the size of a block is limited to 2 MB, and the hash function applied is SHA-256. Measurements are given as the average over 1000 test runs, and the results are presented in **Table 3**. In Surveillant, because the user does not download the block headers of $\mathcal{R}$, but has cross-chain transaction verification through the three-stage Merkle proof (TSMP), the amount of data downloads is significantly reduced. Furthermore, we compare the size of TSMP with the size of ordinary Merkle proof to verify a local transaction in $\mathcal{S}$, as shown in **Fig. 8**. As the height of Merkle mountain range increases logarithmically with the number of the blocks included in it, the TSMP size is kept at about 1.5KB, which is as negligible as the size of ordinary Merkle proof. Comprehensively, the results in **Table 3** and **Fig. 8** show that the cross-chain transaction verification in Surveillant is as nearly efficient as the verification to a local transaction.

Table 3. Bandwidth overhead

	Chain relay	XCLAIM	Surveillant
Data downloaded (MB)	12.1	12.3	1.7

8. Conclusion

To achieve blockchain interoperability, we propose a supervision mechanism between blockchains (Surveillant) in this paper. It introduces the dual-functional nodes to commit the real-time information from $\mathcal{R}$ to $\mathcal{S}$, which enables users to have efficient cross-chain verification. Specially, we introduce Merkle mountain range for blocks aggregation to deal with the large-scale data in $\mathcal{R}$. We propose the design of LOBCounter, which enables the users of $\mathcal{S}$ to retrieve the long orphan branch occurred in $\mathcal{R}$. The incentive mechanism is improved to encourage the behaviors of dual-functional nodes. The security proof and experimental results show that Surveillant is suitable in practice.

References

[1] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Proc. of Annual International Cryptology Conference*, Springer, pp. 357–388, 2017. Article (CrossRef Link)

[2] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *Proc. of 2014 IEEE Symposium on Security and Privacy*, IEEE, pp. 459–474, 2014. Article (CrossRef Link)

[3] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016. Article (CrossRef Link)

[4] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proc. of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 17–30, 2016. Article (CrossRef Link)

[5] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008. Article (CrossRef Link)

[6] V. Buterin et al., "A next-generation smart contract and decentralized application platform," *white paper*, vol. 3, no. 37, 2014. Article (CrossRef Link)

[7] S. Jiang, J. Cao, J. A. McCann, Y. Yang, Y. Liu, X. Wang, Y. Deng, "Privacy-Preserving and Efficient Multi-Keyword Search over Encrypted Data on Blockchain," in *Proc. of IEEE International Conference on Blockchain*, IEEE, pp. 405-410, 2019. Article (CrossRef Link)

[8] Z. Liu, Y. Xiang, J. Shi, P. Gao, H. Wang, X. Xiao, B. Wen, and Y.-C. Hu, "Hyperservice: Interoperability and programmability across heterogeneous blockchains," in *Proc. of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 549–566, 2019. Article (CrossRef Link)

[9] V. Buterin, "Chain interoperability," *R3 Research Paper*, 2016. Article (CrossRef Link)

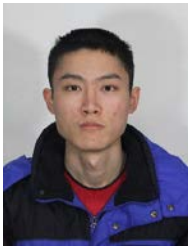
[10] W. Warren, A. Bandeali, "0x: An open protocol for decentralized exchange on the ethereum blockchain," 2017. Article (CrossRef Link)

[11] C. Decker and R. Wattenhofer, "A fast and scalable payment network with bitcoin duplex micropayment channels," in *Proc. of Symposium on Self-Stabilizing Systems*, Springer, pp. 3–18, 2015. Article (CrossRef Link)

[12] M. Herlihy, "Atomic cross-chain swaps," in *Proc. of the 2018 ACM symposium on principles of distributed computing*, pp. 245–254, 2018. Article (CrossRef Link)

[13] "Btc relay," *Ethereum project*, 2015. [Online]. Available: https://github.com/ethereum/btcrelay

[14] A. Zamyatin, D. Harz, J. Lind, P. Panayiotou, A. Gervais, and W. Knottenbelt, "Xclaim: Trustless, interoperable, cryptocurrency-backed assets," in *Proc. of 2019 IEEE Symposium on Security and Privacy (SP)*, IEEE, pp. 193–210, 2019. Article (CrossRef Link)
[15] S. Jiang, J. Cao, H. Wu, Y. Yang, M. Ma, J. He, "BlocHIE: A BLOCkchain-Based Platform for Healthcare Information Exchange," in *Proc. of International Conference on Smart Computing*, IEEE, pp. 49-56, 2018. Article (CrossRef Link)
[16] M. A. Ferrag, M. Derdour, M. Mukherjee, A. Derhab, L. A. Maglaras, H. Janicke, "Blockchain Technologies for the Internet of Things: Research Issues and Challenges," *IEEE Internet of Things Journal*, vol. 6, pp. 2188-2204, 2019. Article (CrossRef Link)
[17] M. A. Ferrag, L. Shu, "The Performance Evaluation of Blockchain-Based Security and Privacy Systems for the Internet of Things: A Tutorial," *IEEE Internet of Things Journal*, vol. 8, pp. 17236-17260, 2021. Article (CrossRef Link)
[18] S. Jiang, J. Cao, H. Wu, Y. Yang, "Fairness-Based Packing of Industrial IoT Data in Permissioned Blockchains," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 11, pp. 7639-7649, 2021. Article (CrossRef Link)
[19] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *Proc. of IEEE P2P 2013 Proceedings*, IEEE, pp. 1–10, 2013. Article (CrossRef Link)
[20] N. Papadis, S. Borst, A. Walid, M. Grissa, and L. Tassiulas, "Stochastic models and wide-area network measurements for blockchain design and analysis," in *Proc. of IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, IEEE, pp. 2546–2554, 2018. Article (CrossRef Link)
[21] L. Kiffer, R. Rajaraman, and A. Shelat, "A better method to analyze blockchain consistency," in *Proc. of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 729–744, 2018. Article (CrossRef Link)
[22] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *Proc. of International conference on financial cryptography and data security*, Springer, pp. 436–454, 2014. Article (CrossRef Link)
[23] J. Bonneau, "Why buy when you can rent?," in *Proc. of International Conference on Financial Cryptography and Data Security*, Springer, pp. 19–26, 2016. Article (CrossRef Link)
[24] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Proc. of Annual international conference on the theory and applications of cryptographic techniques*, Springer, pp. 281–310, 2015. Article (CrossRef Link)
[25] L. Wang and Y. Liu, "Exploring miner evolution in bitcoin network," in *Proc. of International Conference on Passive and Active Network Measurement*, Springer, pp. 290–302, 2015. Article (CrossRef Link)
[26] P. Todd, "Merkle mountain range," Github, 2018. [Online]. Available: https://github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md
[27] G. Wood et al., "Ethereum: A secure decentralised generalized transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014. Article (CrossRef Link)
[28] "Bitcoin development," Github, 2021. [Online]. Available: https://github.com/bitcoin/bitcoin



Xinyu Liang received the master's degree in Computer Science from Central China Normal University in 2018. He is currently working toward the Ph.D. degree in information security, Wuhan University, Wuhan. His research interests include blockchain and computer network.

Jing Chen received the Ph.D. degree in computer science from Huazhong University of Science and Technology, Wuhan. He worked as a full professor in Wuhan University from 2015. His research interests in computer science are in the areas of network security, cloud security. He has published more than 100 research papers in many international journals and conferences, such as TDSC, TIFS, TMC, INFOCOM, TC, TPDS, et al. He acts as a reviewer for many journals and conferences, such as IEEE Transactions on Information Forensics, IEEE Transactions on Computers, IEEE/ACM Transactions on Networking.

Ruiying Du received the BS, MS, PH. D degrees in computer science in 1987, 1994 and 2008, from Wuhan University, Wuhan, China. She is a professor at School of Cyber Science and Engineering, Wuhan University. Her research interests include network security, wireless network, cloud computing and mobile computing. She has published more than 80 research papers in many international journals and conferences, such as IEEE Transactions on Parallel and Distributed System, International Journal of Parallel and Distributed System, INFOCOM, SECON, TrustCom, NSS.

Tianrui Zhao is currently working toward the bachelor degree in information security, Wuhan University, Wuhan, China. His research interests include cryptography and blockchain.