creative commons
COMMONS DEED

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.

변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

Ph.D. DISSERTATION

Addressing Centralization, Security and Performance Issues in Real-World Blockchain Systems

블록체인 시스템의 중앙화, 보안 및 성능 이슈 해결 연구

August 2023

DEPARTMENT OF ELECTRICAL ENGINEERING AND
COMPUTER SCIENCE
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Jongbeen Han

Addressing Centralization, Security and Performance Issues in Real-World Blockchain Systems

블록체인 시스템의 중앙화, 보안 및 성능 이슈 해결 연구

지도교수 엄현상

이 논문을 공학박사 학위논문으로 제출함

2023 년 6 월

서울대학교 대학원

컴퓨터 공학부

한종빈

한종빈의 공학박사 학위논문을 인준함

2023 년 6 월

위 원 장	유 승 주	(인)
부위원장	엄 현 상	(인)
위 원	전 병 곤	(인)
위 원	문 수 묵	(인)
위 원	손 용 석	(인)

Abstract

A blockchain system is engineered to facilitate the execution of transactions in a consistent and dependable manner within an untrusted and decentralized environment. This innovative technology also enables the execution of various contracts without intermediary authorities, due to the capabilities of smart contracts embedded within the blockchain system. The blockchain system can permit the reliable execution of transactions because it distributes all data in a decentralized manner and provides trust through consensus mechanisms.

Nevertheless, in the blockchain system, centralization issues exist. For example, there are problems such as the security vulnerability of the single signature wallet, the centralization issue in the external database when looking up the transaction history in the blockchain, and the centralization tendency of the miner (verifier) of the consensus algorithm.

In this article, we concentrate on solving the centralization issues in blockchain. By addressing these centralization issues, we enhance the security and performance within the blockchain system. There are three key aspects: wallet security, the performance of transaction retrieval, and consensus algorithms.

First, centralized single-signature schemes are insecure. To solve this, we introduce an efficient multi-signature wallet that enhances security, performance, storage efficiency, and privacy without altering the underlying blockchain protocol. By exploiting a threshold elliptic curve digital signature algorithm (T-ECDSA) and bloom filter in the transactions, the proposed wallet demonstrates improved verification performance and reduced transaction size and fee compared to existing wallets.

Next, services using a centralized external database have to rely on their service and have issues with data authority. To solve this, we propose a new scheme to provide SQL query operations within each blockchain node to retrieve the history information of smart contracts and general transactions within the blockchain system. To do this, we combine an embedded relational database in an Ethereum-based blockchain system to provide the SQL query. It enables range query in blockchain without any user-defined data structure and decreases the management cost for the regular transaction without any external database.

Lastly, consensus algorithms tend to be centralized depending on the amount of mining power or staking. To solve this, we propose the Proof of Double Committee (PoDC) consensus mechanism for more decentralization of a block proposer, preventing collusion, and improving performance. In the PoDC, validators are divided into two groups (i.e., standing and steering members), and a coordinator and steering members are randomly selected using double hashing and a Verifiable Random Function (VRF). This approach not only mitigates centralization issues associated with existing consensus algorithms but also can improve blockchain performance than other consensus algorithms, such as Proof of Work (PoW) and Tendermint.

Keywords: Blockchain, Multi-signature, T-ECDSA, Bloom Filter, Delay Function, Zero-knowledge Proof, Double Hashing, Verifiable Random Function (VRF)
Student Number: 2019-38471

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Since the advent of Bitcoin in 2009, blockchain systems have become a significant component in processing transactions between users without the need for a centralized trusted authority. The blockchain systems comprise a large number of non-trusted nodes and utilize a node-to-node consensus algorithm to maintain data integrity [2]. Additionally, to ensure non-repudiation of transactions between users, transactions are signed using the private keys of the involved parties [3].

Blockchain systems are used for several reasons. Primarily, it overcomes the limitations of traditional centralized systems by facilitating reliable transactions in a decentralized environment. For example, the 2008 financial crisis, triggered by the bankruptcy of Lehman Brothers, exposed the vulnerability and unreliability of centralized financial systems. And it led to the emergence of Bitcoin's blockchain as proposed by Satoshi Nakamoto [4]. Public ledgers of blockchain

systems offer transactional transparency, making it challenging for malicious users to manipulate data [4, 5]. As a result, users can securely conduct transactions, and blockchain technology holds the potential to revolutionize numerous industries, including finance, insurance, healthcare, logistics, and real estate.

Nonetheless, blockchain technology is still in a relatively early stage of development and exists centralization issues. The following aspects highlight the need for the technology to evolve further:

- **Addressing security vulnerabilities.** While blockchain inherently ensures data integrity, it remains susceptible to 51% attacks, smart contract vulnerabilities, and wallet security vulnerabilities. Especially in wallets, single-signature wallets are subject to hacking. Consequently, research is necessary to resolve these security issues and enhance the technology's safety [6].

- **Enhancing scalability and efficiency.** Current blockchain systems face limitations in transaction throughput and processing speed [7]. To address this, various research efforts, including fixed validators, new consensus algorithms, layer 2 solutions [8], and sharding [9], are underway.

  In providing transaction histories, most services use an external database to improve service performance. However, due to this way, users have to trust the services that use external databases, which generates another centralization. Ongoing research is essential for solving centralization and improving performance and ensuring their applicability in large-scale systems.

- **Mitigating threats to decentralization.** Blockchain systems have a decentralized structure with multiple participants involved in the network, validating transactions, and creating blocks [10]. The structure

aims to overcome the vulnerabilities and shortcomings of centralized systems. However, centralization may occur in blockchain systems employing the Proof of Work (PoW) algorithm, as large mining pools or mining companies come to dominate the market [11]. The centralization issue in blockchain systems contradicts the blockchain's nature and warrants continuous research and improvement for resolution.

Research on enhancing the performance and security of blockchains has a significant interest within the blockchain community, leading to numerous papers being published at recent conferences. This article seeks to offer new insights by exploring blockchain systems and addressing the fundamental challenges of blockchain systems.

1.1.1 Problems and Approaches

There are several challenges associated with blockchain systems. The followings are the three problems to be addressed in this dissertation:

Security and performance of blockchain wallet One of the blockchain features is a digital signature that provides non-repudiation. And, blockchain wallets support digital signatures via using keys (i.e., private and public keys). The keys are used to sign a transaction and verify the identity of the transaction signer. Depending on the number of keys used, there are two types of digital signature schemes: single-signature and multi-signature schemes. A single-signature scheme is a digital signature scheme that allows only one user to agree to a transaction by their own sign. Therefore, this scheme can be vulnerable to security attacks due to a single-point attack. By contrast, a multi-signature scheme is a digital signature scheme that allows a group of users to agree to a transaction by their sign [12]. Therefore, the multi-signature scheme provides

higher security than a single-signature scheme by distributing the authority for signing a transaction. However, the multi-signature usually generates multiple signatures that are to be validated and includes all participants' public keys in a transaction for validating the transaction. And it shows a lower performance for validating signed transactions and larger transaction sizes since multiple signatures are to be made and validated.

Previous studies [13, 14] show the result of investigating multi-signature on blockchains to enhance the blockchain wallet's security. BitGo [14] proposed pay-to-script hash (P2SH), which is a new type of Bitcoin address introduced as part of Bitcoin improvement proposal (BIP) 16 for supporting multi-signature on Bitcoin. MultiSigWallet [13] provides high security by allowing multiple parties to agree on transactions based on a smart contract. The study reported herein uses the approaches [13,14] in terms of investigating a blockchain wallet. We focus on providing high-performance multi-signature and storage efficiency on blockchains by reducing the number of transactions to be validated and the transaction size.

Search performance issue Most blockchain systems are built using LevelDB. The LevelDB is a sort of key-value store and provides great performance in sequential reads and writes. However, the LevelDB (i.e., key-value database) does not support SQL query operations such as range queries. Furthermore, LevelDB provides low retrieval performance for a large amount of data. Therefore, to handle these issues, existing systems use user-defined data structures for a smart contract and an external relational database to retrieve a range of data [15, 16]. However, user-defined data structures in a smart contract can decrease the overall performance. Also, an external database can increase management costs.

To provide higher search performance in the blockchain system, the previ-

ous studies [15, 17, 18] show the result of improving the select query operation. Etherscan [15] supports the select query operation that retrieves blockchain information such as transactions, addresses, tokens, prices, and other activities using an external database system. Pratama et al. [17] have let users and developers easily access blockchain data by adding three main query functions (retrieval query, aggregate query, and aggregate query). The Graph [18] provides services of querying from blockchain data by continually scanning all of the events. Our study is in inline with these studies [15, 17, 18] in terms of inspecting the search performance issues in a blockchain system. In contrast, to improve search performance and reduce management costs while maintaining decentralization, we concentrate on retrieving range data of both smart contracts and regular transactions using an embedded relational database in a blockchain system.

Centralization and Performance Degradation Issues by Consensus Algorithms One of the important blockchain features is a consensus algorithm which is designed to ensure the accuracy and consistency of the transaction data [19]. Each blockchain has a different consensus algorithm to provide data integrity and high performance while maintaining decentralization. For example, Bitcoin and Ethereum blockchain use a proof of work (PoW) consensus algorithm that the miner who first solves the crypto puzzle has a block generation authority [4]. And, in the case of Cosmos blockchain, which is using the Tendermint consensus algorithm, a proposer who is among the validators has the block generation authority [20]. And, only validators who are selected according to the staking and delegated amount of digital assets can participate in the consensus to validate a block. Therefore, for gaining the authority of creating blocks, users need huge computing power to solve the crypto puzzle in PoW or need many digital assets (e.g., ATOM) to get vote power in the Ten-

dermint. It means that a user who has more economically superior power has more chance to generate a block and vote a block, and it brings out to increase the likelihood of centralization.

To solve centralization issues in the blockchain, the previous studies [21, 22] show the result of investigating validators and proposer selection on blockchains consensus algorithm for providing security. Alzahrani et al. [21] proposed a new true decentralized consensus protocol utilizing game theory and randomness. They mainly address the problem of validators' selection in terms of how to select them and how many to select to achieve a satisfactory trade-off between security and efficiency. Additionally, their protocol selects the feature offered by game theory to reward honest adhered parties and punish malicious ones.

Gilad et al. [22] proposed a new Byzantine Agreement (BA) protocol mechanism based on Verifiable Random Functions (VRF). The protocol allows users to privately check whether they are selected to participate in the BA.

Our study is in line with those approaches [21, 22] in terms of investigating a consensus algorithm for random selection of validators. In contrast, we focus on distributing the validator set as two groups (i.e., standing committee and steering committee) for complementing the disadvantage of preventing unity and providing high performance.

1.2 Contributions

In this dissertation, our contributions are summarized as follows:

- We have analyzed the security and performance of existing blockchain wallets. And we propose a new multi-signature wallet that shows better performance, storage efficiency, and privacy than existing blockchain wallets; the proposed wallet involves T-ECDSA and a Bloom filter and does

not require any modification of the blockchain protocol. We experimentally demonstrate that the proposed multi-signature wallet shows better transaction validation performance, storage efficiency, and cost efficiency compared with existing multi-signature wallets.

- We have investigated the performance of existing blockchain systems with search operations. And we propose a scheme that enables SQL query processing to provide decentralization, decrease the management overhead and increase search performance on a blockchain. In addition, we show that the proposed system improves the search performance of smart contract data and reduces the management cost for regular transactions without any external database.

- We have investigated the existing consensus algorithm in terms of both decentralization and performance. We propose a new consensus algorithm (PoDC) that enables the secure selection of a coordinator and validators via double hashing and verifiable random function (VRF) on a blockchain system. The proposed consensus algorithm shows more decentralized and better performance compared with the existing consensus algorithm.

1.3 Outline

This dissertation is structured as follows:

- **Chapter 2** covers the background of blockchain systems.

- **Chapter 3** introduces `new efficient multi-signature wallet`, which can provide high validation performance on blockchains owing to the advantage of using a single signature and can identify each transaction's participant with a small transaction size. We describe the details of the

design and implementation of our scheme and evaluate our scheme on the Bitcoin and Ethereum blockchain.

- **Chapter 4** introduces `an enable SQL query processing` with combining embedded database systems and providing register and query mechanism. We start by explaining the problems of existing retrieval of historical data in the blockchain and analyze the root cause of searching problems. And we give details of how we can address the challenge and evaluate our scheme compared to existing mechanisms in the blockchain.

- **Chapter 5** introduces `Proof of Double Committee (PoDC) consensus algorithm` by separating validators into two groups and selecting the validators via double hashing and verifiable random function (VRF) for improving the security and performance of the blockchain. We start by explaining the problems of existing consensus algorithms in blockchain systems and analyze the decentralization of the blockchain and its performance. And we give details of how we can address the challenge and evaluate our scheme compared to existing consensus algorithms (e.g., bitcoin, tendermint, etc.).

- **Chapter 6** summarizes and concludes the dissertation. It also points out directions for future work.

Chapter 2

Background

2.1 Blockchain

Blockchain is a distributed storage technology that records data transparently and replicates it across multiple nodes, creating a public ledger accessible to anyone. Additionally, blockchain serves as a decentralized infrastructure and distributed computing system that employs consensus algorithms and chain data structures to verify data integrity. As a result, users can distribute data through blockchain technology, maintain data integrity, support Byzantine fault tolerance (BFT), and transfer digital assets without intermediaries [23].

A blockchain consists of transactions, blocks, and the blockchain itself. A transaction represents data that alters the state value associated with a wallet or smart contract address on the blockchain. Users sign transactions with their private keys to approve state changes, and these transactions are validated using a public key mapped to the private key. Once a transaction is validated, it can be included in a block. A block, the fundamental unit of a blockchain, contains

information such as transaction data, block hash, the previous block's hash, a timestamp, a nonce, etc. Transaction data is stored in the order in which transactions occur within a block, while the block hash serves as a unique identifier for the block [24]. The previous block's hash is used to connect blocks into a chain, ensuring the data integrity of the blockchain. A blockchain represents a continuous chain of blocks, with each current block linked to the previous one based on the previous block's hash value [25].

2.2 Blockchain Wallet

2.2.1 Single-signature wallet

A blockchain wallet is a financial application at provides an interface for managing digital assets (e.g., Bitcoin, Ethereum, etc.). It manages a user's private and public keys for signing a transaction involving the transfer of digital assets and execution of a smart contract. A user can prove that he/she owns the digital assets on the blockchain by signing a transaction with the private key in the wallet. Most blockchain wallets manage digital assets using a single-signature scheme such as the elliptic curve digital signature algorithm (ECDSA) [26]. This algorithm is a variant of the digital signature algorithm (DSA) by using elliptic curve cryptography (ECC). ECC is a public-key cryptography approach that is based on the algebraic structure of elliptic curves over finite fields [27].

The ECDSA is more secure than the Rivest–Shamir–Adleman (RSA) [28] algorithm against current cracking methods such as brute-force attacks owing to its complexity. Consequently, a blockchain wallet based on the ECDSA can sign a transaction with a higher security level and a relatively small key size. However, the blockchain wallet has security issues since the private key is susceptible to attack from a single attack point because of the nature of the

ECDSA. Therefore, a more secure mechanism is required for blockchain wallets.

2.2.2 Multi-signature wallet

To compensate for the disadvantage of the single-signature wallet, the multi-signature wallet allows a group of users to agree to a transaction by their signs and divides up responsibility for the transmission of digital assets and execution of a smart contract among multiple users [12]. Participants signing a transaction can be identified from the participant's public key in the multi-signature wallet. This multi-signature wallet can provide higher security for blockchain transactions since it distributes the authority of the wallet to several participants. For example, in the case of a single-signature wallet, if only a private key of the wallet is obtained by a malicious attacker, the digital assets can be stolen. Meanwhile, a multi-signature wallet provides higher security than a single-signature wallet because a malicious attacker has to acquire a certain number of private keys; this number of keys is configured when the wallet is generated.

However, the multi-signature wallet has some issues that increase the transaction validation and the transaction size in blockchains. For example, in the Bitcoin blockchain, a multi-signature wallet signs a transaction while generating multiple signatures, and the signed transaction is verified by the full nodes on the blockchain network. Therefore, it increases the verification time compared with that of a single-signature scheme because multiple signatures should be verified. Moreover, when the multi-signature wallet is used on a blockchain, the blockchain protocol should be modified [29]. Meanwhile, in the case of Ethereum, a smart contract should be used for utilizing the multi-signature wallet [13].

2.3 Regular transaction in blockchain system

A regular transaction in the blockchain system is cryptographically signed instructions from accounts. An account will initiate a transaction to update the state of the account in the blockchain network. Thus, a regular transaction is an act of transferring digital assets from one wallet address to another wallet address. For example, in the Ethereum blockchain, if Bob sends Alice 1 eth, Bob's account must be debited and Alice's must be credited. This state-changing action takes place by a transaction [30]. The submitted regular transaction includes the following information: recipient, signature, amount of eth to transfer from sender to recipient data, gas limit, the maximum amount of gas, etc.

2.4 Smart contract in blockchain system

A smart contract is an important component in the Ethereum blockchain system that was initially proposed by Vitalik Buterin [31]. Smart contracts are carried out correctly on the blockchain system guaranteed by consensus protocol. [32]. In addition, a smart contract can encode any set of rules represented in its programming language (e.g., solidity). A smart contract, for example, may make transfers and apply various business logic, including financial instruments, insurance, real estate, and medical, among other things, on the blockchain.

2.5 Key-Value Store

An Ethereum blockchain system maintains three tries (i.e., world state trie, transaction trie, and transaction receipts trie) [33,34]. The Ethereum blockchain system's *world state trie* includes the states of users and smart contracts; the *transaction trie* records transactions that alter states of users and smart contracts; and the *transaction receipts trie* keeps the results of completed transac-

tions. These three tries are stored in a key-value database known as a sort of database designed for storing, retrieving, and managing data.(i.e., LevelDB [35]). The key-value store performs well in sequential reads and writes and offers quick read and write operations for each key. Yet, the operation of several keys in the key-value store can be slow because the key-value store does not provide range query operations without the relational data model.

2.6 Consensus Algorithm

A blockchain consensus algorithm is a set of rules and procedures by which participants in a blockchain network agree on the validity of transactions and blocks. Consensus algorithms play a crucial role in ensuring the security, stability, and decentralized nature of blockchains. Various consensus algorithms exist, each with specific goals and use cases [36].

Proof of Work (PoW) The first consensus algorithm used in many blockch ain networks, including Bitcoin. In PoW, participants solve complex mathematical problems to create blocks and receive rewards proportional to the amount of work they contribute to the network. PoW provides high security but has the disadvantage of very high energy consumption and can have centralization issues by the mining pool [37].

Tendermint Tendermint is a consensus algorithm based on PBFT, providing high throughput and short block times. However, there is a risk of centralization in the process of electing voters, and it may be vulnerable to Byzantine attacks. Tendermint works best in situations where trust exists between participants, and performance can suffer in situations where trust is lacking [20].

Proof of Stake (PoS) As a consensus algorithm developed to reduce energy consumption, participants gain block creation rights proportional to

their token stake. PoS is energy-efficient and can alleviate centralization issues. Ethereum is transitioning to Ethereum 2.0, a PoS-based consensus algorithm [38].

Practical Byzantine Fault Tolerance (PBFT) An algorithm that achie ves consensus while tolerating Byzantine faults in a distributed system. Participants send and receive messages to each other to verify the validity of blocks and reach a consensus. PBFT offers high throughput and low latency but is only effective in situations where reliability is required like private blockchain [39].

Delegated Proof of Stake (DPoS) As a consensus algorithm that improves on PoS, participants delegate their stake to specific nodes to give them the right to create blocks. Delegated nodes are responsible for block generation and validation, achieving high throughput and low latency [40]. However, DPoS has also been criticized for allowing delegated nodes to exercise too much power.

The research and development of consensus algorithms play a crucial role in the advancement of blockchain technology. As a result, further investigation into blockchain consensus algorithms is necessary, and it is anticipated that the security, scalability, fairness, and energy efficiency of blockchain networks will continue to improve.

2.7 Related Work

In several academic fields, blockchain presents many challenges. In terms of wallet security, there are several research. First, to increase security in digital signature schemes, previous studies [41–43] show the result of investigating the threshold signature scheme for improving the signature performance. Gennaro et al. [43] proposed the first protocol that supports a multi-party threshold

ECDSA with an efficient distributed key generation. Thus, it reduces the execution time and data transfer time for key generation. Goldfeder et al. [41] introduced the first practical t-of-n threshold signature scheme compatible with Bitcoin's ECDSA signatures. They show how Bitcoin wallets use a threshold signature scheme for signing and verifying a transaction. Gennaro et al. [42] introduced a threshold-optimal digital signature algorithm (DSA) scheme for minimizing the number of servers targeted by a malicious attacker. Our study is in line with existing approaches [41–43] in terms of exploiting the threshold signature in the blockchain wallet system to decrease the validation and signature times. However, we furthermore reduce the transaction size and improve privacy by using a bloom filter apart from improving the validation and signature performance.

In addition, to keep crypto funds safe, studies show the result of investigating hardware wallets. Hardware wallets (e.g., Ledger [44], Trezor [45], and Keepkey [46]) allow storing the private key in offline and secure storage. They can operate directly on a personal computer or mobile phone through Bluetooth or by being plugged in via USB instead of the Internet. Thus, the private keys would remain completely secure even if the hardware wallet connects to a computer infected with a virus. In addition, for higher security, a multi-signature provides multiple keys to authorize a Bitcoin transaction, rather than a single signature from one key. Our study is in line with these approaches [44–46] in terms of improving the security of wallets. Furthermore, we improve higher security by distributing the authorities of the wallets against a single-point attack.

Also, there have been studies on improving the security of blockchain wallets via distributing the authorities. MultiSigWallet [13] provides an m-out-of-n multi-signature wallet with smart contracts on Ethereum. It executes basically

smart contracts able to store tokens and requires that the issued transaction needs to accept some requirements, including a specific signing configuration. BitGo [14] proposed a 2-of-3 multi-signature wallet and a new type of Bitcoin address introduced as part of BIP 16 (Bitcoin Improvement Proposal 16). It provides more secure Bitcoin addresses compared with traditional Bitcoin addresses by using the multi-signature wallet. Our study is in line with these approaches [13, 14, 44–46] in terms of improving the security of wallets. In particular, we focus on providing higher performance for multi-signature on blockchains by reducing the number of transactions to be validated and the transaction size.

In terms of the performance of retrieval procedures in blockchain, there are several research. **Systems using an external database.** To increase retrieval performance, previous systems [15, 47, 48] use an external database for making an indirect query in a blockchain system. Etherscan [15], a search, API, and analytics platform for Ethereum, is one of the attempts to use an external database. It enables users to browse the Ethereum blockchain in search of transactions, addresses, tokens, prices, and other Ethereum-related activity. Etherchain [47] is an explorer for the Ethereum blockchain by extending the Ethereum native API. It offers basic statistical information like transaction count and block time. Additionally, it enables users to investigate smart contract transactions, search for transactions, and monitor user account balances. Ethstats [48] is a visual interface for monitoring the Ethereum network's health, and it offers the most recent data on a block number, connected node information, pending transactions, gas prices, etc. FlureeDB [49], and BigchainDB [50] provide developing blockchain database solutions to support SQL-like queries. However, in the case of those systems (e.g., Etherchain [47], EtherScan [15], and Ethstats [48]), users cannot verify whether the results of the external database and the blockchain

data are identical since they use external database outside of the blockchain system. Also, External database systems have significant maintenance costs to manage.

Embedded Blockchain systems. There are other ways to increase retrieval performance, previous studies [17, 51–54] proposed a new layer and new language for the blockchain. To increase select query efficiency, for instance, EtherQL [51] added a querying layer to the Ethereum client. It offers a variety of queries, such as top K queries and range queries for transactions and blocks. VQL [53] provided efficient query services by extracting transactions from the underlying blockchain system and adding a middleware layer. EQL [54] proposed a query language that can retrieve information from the blockchain written in the programming language smalltalk [55]. And, through this query language, users can get block and transaction data. Pratama et al. [17] added three main query functions (retrieval query, aggregate query, and aggregate query) so that users and developers can easily access blockchain data.

For users or analysts who often generate transactions, these earlier systems and studies enhance the performance of searching for information about Ethereum's blocks, transactions, and accounts. Our study is in line with these studies in terms of investigating the performance of select operations in the blockchain system. On the other hand, our work focuses on decentralization and obtaining a range of data from both smart contracts and transactions utilizing an embedded relational database in a blockchain system that is based on Ethereum.

In terms of decentralization, previous research [33, 56] has explored consensus algorithms, a collection of mechanisms designed to address the problem of centralization. Bitcoin [56] introduced the proof of work consensus algorithm to tackle this issue, leveraging the concept of decentralization by enabling net-

work participants to competitively undertake computational tasks. These tasks allow the generation of blocks encompassing new transactions, eventually leading to a consensus. The purpose of the proof of work (PoW) consensus algorithm is to achieve decentralization for digital asset transactions. Moreover, Ethereum [33], utilizing both Proof of Work and Proof of Stake algorithms, facilitates the transfer of data ownership from corporations to users in existing centralized systems via smart contracts. However, Mauro et al. [57] highlighted that although blockchains based on the PoW consensus algorithm are designed to achieve network-wide consensus, certain participants can wield undue influence, thereby undermining decentralization due to economic advantages during the process. Our study aligns with these investigations in its exploration of the centralization problem within blockchain systems. However, we further concentrate on resolving the issue of disproportionate influence exerted by certain participants by introducing a random selection method for block proposers using double hashing.

Previous studies [22, 58, 59] have investigated the consensus algorithm with fixed validators and validator selection algorithm for improving performance in the blockchain. Cosmos [58] proposed the Tendermint BFT(Byzantine Fault Tolerance) consensus algorithm for selecting block validators and generating blocks. This algorithm operates on a Proof of Stake (PoS) mechanism, where validators are chosen based on the quantity of the token, ATOM, they have staked. Currently, the top 150 staking participants are selected as block validators. Gilad et al. [22] proposed a new Byzantine Agreement (BA) consensus algorithm based on Verifiable Random Functions (VRF). In particular, Algorand employs the Pure Proof of State(PPoS) mechanism, where users are randomly chosen as block validators in proportion to the quantity of Algorand tokens (Algos) they hold. EOS [59] exploits Delegated Proof of State (DPoS)

consensus algorithm. EOS users nominate 21 block producers on the network who are responsible for validating all transactions and generating the blocks. This selection process occurs through the votes of EOS token holders, with the voting power increasing proportionally to the number of tokens held. Our study is in line with these studies in terms of investigating to improve performance via fixed the number of validators in the blockchain system. On the other hand, our work additionally focuses on allowing all nodes to participate in the network and fairly distributing block generation rights.

Chapter 3

Efficient and Secure Multi-Signature Wallet

3.1 Motivation

3.1.1 Blockchain wallet

In the rapidly evolving world of blockchain technology, secure management of digital assets has become a critical concern for users. One of the key components of a blockchain system is the wallet, which enables users to store and manage their digital assets. However, various instances of hacking and theft have exposed the vulnerabilities in single-signature wallets, highlighting the need for more secure alternatives.

For example, In 2018, when the Japanese cryptocurrency exchange Coincheck was hacked, resulting in the loss of approximately 530 million dollars worth of NEM tokens [60]. In September 2020, KuCoin experienced a massive breach, losing approximately 280 million dollars in funds, which included Ethereum and ERC20 tokens [61]. In November 2019, Upbit faced a similar

predicament when hackers made off with 50 million dollars of Ethereum [62]. In May 2019, Binance faced a hack where around 40 million dollars was stolen [63]. The hackers exploited the exchange's usage of single-signature wallets, which rely on a single private key for transaction authentication. The theft demonstrated the inherent security weaknesses in single-signature wallets and emphasized the necessity of developing more robust wallet security measures to protect user assets.

Therefore, cryptocurrency exchanges employ a multitude of strategies to safeguard their customers' assets, typically employing a dual structure of "Hot Wallets" and "Cold Wallets." Hot Wallets are online, connected to the internet, and facilitate real-time transactions. They allow exchanges to expedite the processing of user deposit and withdrawal requests. Nevertheless, their internet connectivity also renders them susceptible to hacking threats. Cold Wallets, on the other hand, are disconnected from the internet, offering a much higher level of security. Most cryptocurrency exchanges store the majority of their customer assets in these cold wallets to ensure their safety. However, even in this way, Hotwallet is still subject to hacking. In addition to these measures, many exchanges adopt multi-signature wallets to enhance security, but it is impossible to use multi-signature wallets unless the blockchain protocol supports multi-signature methods. To compensate for this, a multi-signature method is programmed and used in a blockchain where smart contracts can be written.

The background and motivation of this research article stem from such real-world cases of hacking and the increasing demand for security solutions in the blockchain ecosystem. By exploring the limitations of single-signature wallets and examining the potential of multi-signature wallets, we aim to develop an efficient and secure wallet solution that mitigates risks, enhances storage efficiency, and ensures user privacy. This research not only addresses the current

challenges faced by wallet users but also contributes to the broader goal of fostering trust and confidence in blockchain technology as a reliable platform for digital asset management.

3.1.2 Threshold signature scheme

A threshold signature scheme (e.g., T-ECDSA and Shamir's secret sharing scheme) enables n participants to share the power to generate a digital signature with a single public key. A threshold t is specified such that any subset of $t + 1$ participants can jointly sign, but any smaller subset cannot [43,64]. For example, the T-ECDSA generates a single signature from all the signatures of $t + 1$ participants while Shamir's secret sharing scheme generates one signature by using a private key reconstructed with $t + 1$ shares of participants centrally. Both threshold signature schemes produce a signature that is validated with a common public key and they are compatible with an existing ECDSA that does not require the modification of the blockchain protocol.

Generally, when signing a transaction, the ECDSA generates a signature by using the generation point of an elliptic curve G, a random number k, a hash value of message (e.g., transaction) z, and a private key pk [26]. As shown in the equation 3.1, first, the ECDSA generates a point P_{xy} by multiplying the generation point of elliptic curve G and the random number k. The r value is used to generate the s value, and both r and s values are used to validate the transaction. The r value is calculated using the x coordinate of point P (P_x) by modulo n. The parameter n is the integer order of G and is a large prime number. The s is a value generated by a hash value of message (e.g., transaction) z, random k, private key pk and r by a single user. In the case of the T-ECDSA, k and pk are decided by multiple users rather than a single user, without exposing the user information. For example, the k and pk values can

be calculated via shared keys (i.e., k_1/pk_1, k_2/pk_2, ..., k_n/pk_t) between users (i.e., $user_1$, $user_2$, ..., $user_n$). Therefore, this algorithm provides higher security than the ECDSA since it distributes the signing authority.

$$\begin{aligned} P_{xy} = k \cdot G, r = P_x \ mod \ n \\ s = k^{-1}(z + rpk) \ mod \ n \end{aligned} \tag{3.1}$$

Furthermore, the security of the T-ECDSA is identical to that of a multi-signature scheme despite the use of a single signature since it generates several shared keys. Additionally, the verification algorithm of the T-ECDSA is identical to that of a single signature scheme the T-ECDSA generates a single signature. Therefore, we use the T-ECDSA on a blockchain to provide better security and verification performance compared with single-signature and multi-signature schemes, respectively.

3.1.3 Bloom filter

A bloom filter is a space-efficient probabilistic data structure. It is used to validate whether an element is a member of a set [65]. One of the bloom filter features is a false-positive resulting from a hash collision. Therefore, a query of the bloom filter returns the information that an element is probably in a set or that it is definitely not in a set. A bloom filter consists of an array structure of size m bits and uses k different hash functions. A bloom filter size can be decided with an array of m-bits and the k hash functions by using an equation 3.2.

$$\begin{aligned} m = ceil((N * log(P))/log(1/pow(2, log(2)))) \\ k = round((m/N) * log(2)) \end{aligned} \tag{3.2}$$

N is the maximum number of items added to the filter, and P is the probability of false positives. A bloom filter involves two processes: adding an element

to the filter and validating whether the element is in a set or not. To add an element, a bloom filter uses the k hash functions to obtain the k array positions. The bloom filter then sets the bit value at all these positions to one on an array of size m bits. To validate whether an element is in a set, the bloom filter uses each of the k hash functions to obtain the k array positions. If the bit at any of these positions is zero, the element is definitely not in the set; otherwise, it is probably in the set.

We use a bloom filter to identify participants in a signed transaction. By storing the bloom filter instead of a participant's information (i.e., public key or address) on the blockchain, we can provide higher confidence and reduce the transaction size[1].

3.2 Design and Implementation

To achieve higher performance, storage efficiency, and privacy, we propose a new efficient multi-signature wallet. We aim to increase security compared with a single-signature wallet and improve performance and storage efficiency compared with a multi-signature wallet. To do this, to improve the performance (i.e., transaction validation), we devise a blockchain wallet by exploiting a threshold elliptic curve digital signature algorithm (T-ECDSA) [43]. With T-ECDSA, our proposed wallet can generate a single signature by reconstructing the participant's multiple signatures and validating the generated single signature on blockchains. It provides better validation performance since a single signature validation performance is better than that of multiple signatures.

Second, we exploit a Bloom-filter [65] on a transaction to enhance privacy and storage efficiency. With the Bloom filter, we could identify each participant

[1]Generally, the size of a public key is 64 bytes and the size of an address is 20 bytes. Meanwhile, the size of the bloom filter is up to 36 bytes

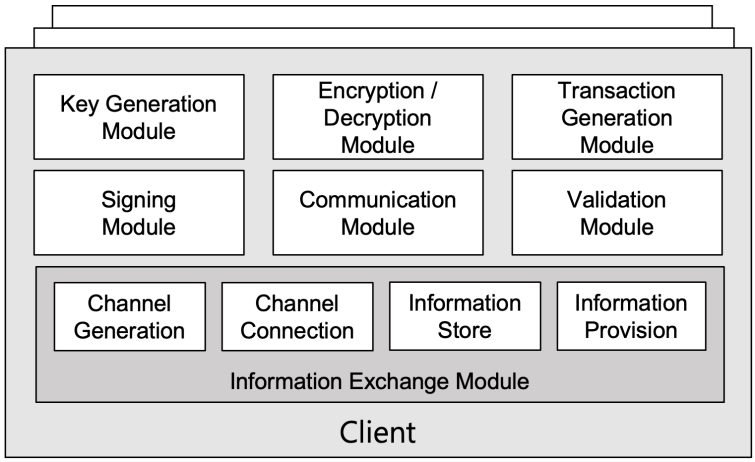

Figure 3.1: Overall architecture of proposed scheme

of the single signature generated by the T-ECDSA. This scheme can increase storage efficiency by storing the Bloom filter instead of public keys in a signed transaction or addresses (the size of public keys and addresses are larger than that of the Bloom filter). This scheme can also prevent the exposure of personal information since the Bloom filter does not store actual data. Finally, our proposed multi-signature wallet can be used in any blockchain (e.g., Bitcoin, Ethereum) without modifying the blockchain protocol or using a smart contract.

3.2.1 Overview

Figure 3.1 shows the overall architecture of the proposed multi-signature wallet which comprises clients (i.e., blockchain wallets).

The client supports managing private/public keys and signs a transaction. To facilitate this, we devise seven modules in the client as shown in Figure 3.1.

The *information exchange module* supports the exchange of each client's information such as public key, encryption key, and local reconstruction of the

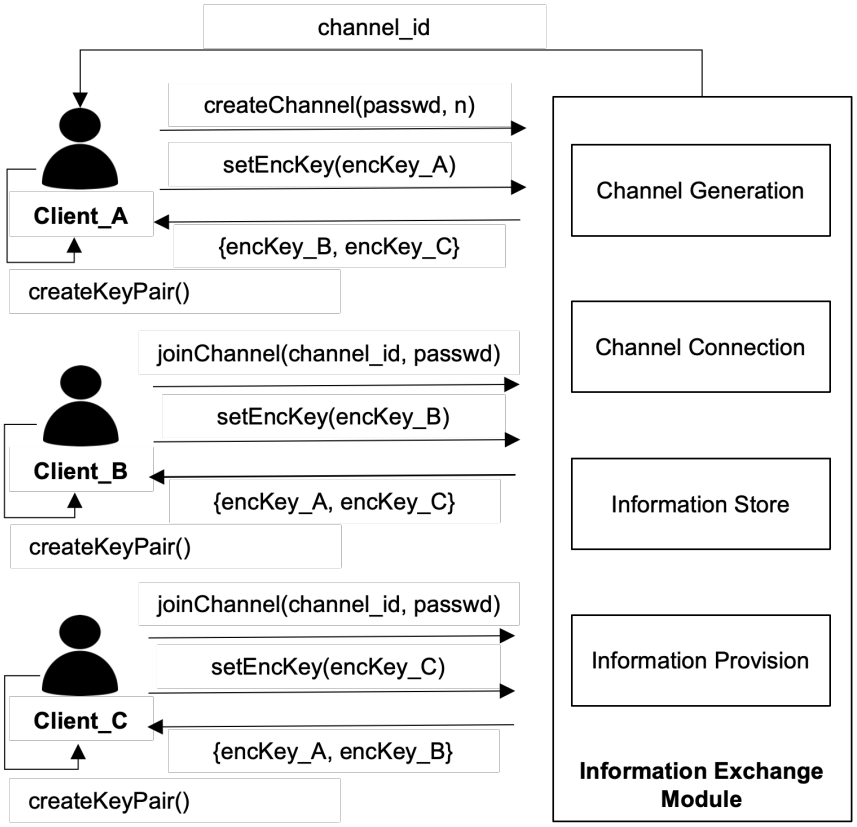

Figure 3.2: A pre-processing overview

participants' multiple signatures and validation signature. To do this, we devise four functions such as *channel generation*, *channel connection*, *information store*, and *information provision* in the module. The *channel generation* creates a channel for client communication. The *channel connection* allows clients to connect to the created channel to exchange information. For example, the clients can generate a common public key of a multi-signature wallet via the channel[2]. The *information store* stores the information generated by each client and the *information provision* transfers the participants' stored information to

[2]The common public key becomes the representative address of a multi-signature wallet. The address is used to receive and send digital assets on the blockchain network [66].

the clients. Note that all information stored is encrypted, protecting the information against a malicious client.

The *key generation module* generates private and public keys. It also creates a common public key of a multi-signature wallet by recombining all participant's public keys. The *encryption and decryption module* performs encryption and decryption to prevent the disclosure of the participants' information. The *transaction generation module* generates a blockchain transaction (e.g., Bitcoin and Ethereum). Also, it creates and stores a bloom filter that represents participants signing a transaction.

The *signing module* signs a transaction by generating a local signature per participant for the transaction and reconstructing the local signatures. This module generates a single signature from multiple local signatures. The *communication module* connects the *information exchange module* to the store or extracts information on clients. The *validation module* verifies whether the client information is forged. If the information is forged, the module stops the process of generating a multi-signature wallet or signing a transaction. Furthermore, this module identifies participants of a transaction by using the Bloom filter.

3.2.2 Preparing to exchange information.

Before generating a multi-signature wallet and signing a transaction, clients perform a preprocessing step for secure information exchange. In this step, the clients connect to a *information exchange module* and exchange their own encryption keys with each other. Figure 3.2 shows the preprocessing step. First, client A requests to create and join a channel with a channel's password (`passwd`) and the number of wallet participants (`n`) to the *information exchange module* via `createChannel()`. The *channel generation module* in the *information exchange module* generates a channel ID (`channel_id`) with the

received password. It returns the channel ID to the client[3]. After then, the rest of the clients (clients B and C) can join the channel with the channel ID and password via `joinChannel()`.

After all, clients join the channel, each client generates an encryption key (`encKey`) / decryption key (`decKey`) pair via `createKeyPair()` and, uploads the encryption key to the *information exchange module* via `setEncKey()`. The *information store module* stores and manages the encryption key. And then, each client requests the other client's encryption keys to the *information provision module*, and the module provides the encryption keys to clients. The encryption key is used to encrypt the information when generating a multi-signature wallet, for high security. To verify whether the encryption key is correct or not, we use zero-knowledge proof [67], which is a method that allows data to be verified without revealing that data. If the proofs are not valid, the key generation and signing process is stopped. This preprocessing guarantees that a multi-signature is generated and a transaction is signed securely.

3.2.3 Generating a multi-signature wallet

To transfer digital assets on a blockchain, the clients create a multi-signature wallet. In our scheme, the multi-signature wallet operates as a single-signature wallet to reduce the validation time. To do this, we generate a common public key and shared keys as follows. As mentioned in Section 3.2.2, clients perform a preprocessing step before generating a multi-signature wallet. After the step, all the clients generate private/public keys via the *key generation module* and share and bind the public key among themselves by using a commitment scheme via the *communication module*. A commitment scheme is a cryptographic primitive that allows one to commit to a chosen value (i.e., public key) while keeping it

[3]The channel ID and password will be shared between clients offline

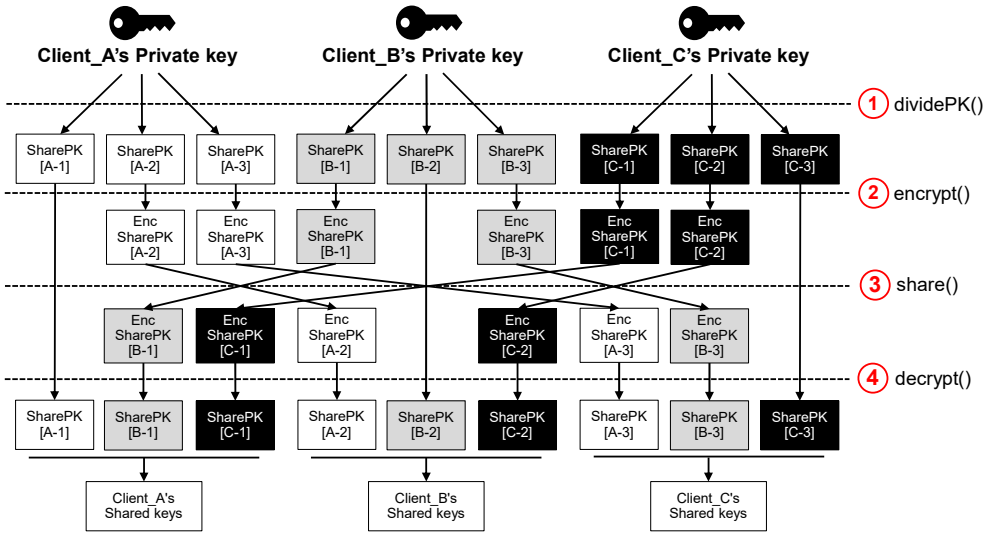

Figure 3.3: A key generation process for 2 of 3 multi-signature wallets

hidden from others, with the ability to reveal the committed value later [68]. If all the public keys are valid, each client generates a common public key with the public keys. The common public key represents the multi-signature wallet and is generated by the *key generation module*.

To generate shared keys, each client divides its own private key to N shared keys with the threshold (T) and the number of participants (N). For example, Figure 3.3 shows a 2 of 3 multi-signature wallet generating shared keys of a multi-signature wallet. Each client (i.e., client A, client B, and client C) first splits its own private key as three shared keys (SharedPK) via `dividePK()` in the *key generation module* (①). The shared keys are used to sign a transaction. Second, each client encrypts the shared keys with participants' encryption keys via `encrypt()` in the *encryption module* (②). For example, client A encrypts the shared key (sharedPK [A-2]) to be transferred to client B with client B's encryption key. Similarly, the shared key (sharedPK [A-3]) to be transferred to client C is encrypted client C's encryption key. The encrypted shared key [A-2]

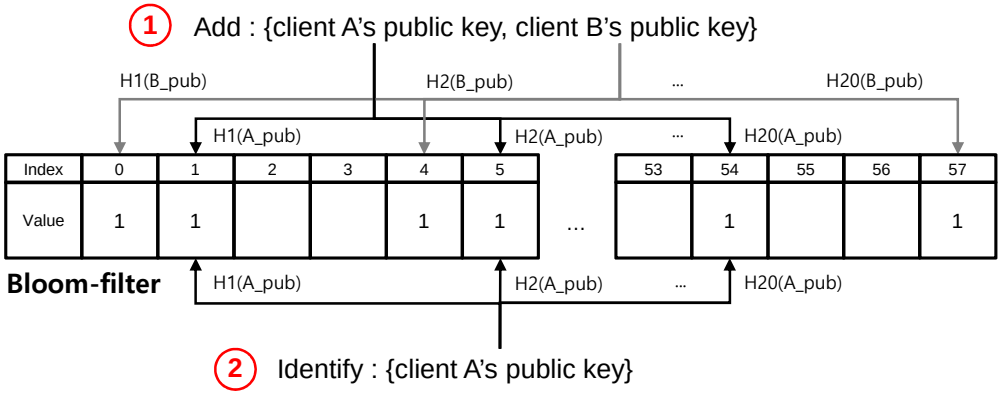

Figure 3.4: Using bloom-filter in a multi-signature wallet (H: hash function, pub: public key)

(enc sharedPK [A-2]) can be opened only by client B, and the encrypted shared key [A-3] (enc sharedPK [A-3]) can be opened only by client C.

After the encryption, each client distributes encrypted shared keys to their associated clients via `share()` in the *communication module* (③). Next, the clients decrypt the transferred shared keys with their decryption keys via `decrypt()` in the *decryption module* (④). Note that the clients exchange each shared key by using zero-knowledge proof (ZKP). Therefore, all clients verify whether the shared keys are valid without actually transferring the shared keys via *validation module*. If the proofs are not valid, the key generation process is stopped.

3.2.4 Signing transaction via multi-signature and Bloom-filter

As mentioned in Section 3.2.2, clients perform a preprocessing step before signing a transaction using multi-signature. After the preprocessing step, to identify the participant of a transaction, clients generate a bloom filter with the participants' public keys via the *transaction generation module*. As mentioned Section in 2.4, because the bloom filter has a false-positive feature, all clients should decide on a bloom filter size with an array of m-bits and k hash functions. We

calculate the m-bits size and the number of k hash functions by using an equation 3.2, the fixed number of participants, and a false positive rate. We set the probabilistically safe state as 2^{-20} and the maximum number of participants in a transaction as ten. As a result, the bloom filter has from 29 bits up to 289 bits of size and 20 hash functions, which is the optimal number of the bloom filter.

Figure 3.4 shows generating a bloom filter of a transaction when two clients (clients A and B) participate in a signing process of a 2 of 3 multi-signature wallet. First, each client calculates m-bits size with the number of participants (i.e., 2). In this case, the Bloom-filter size is 58 bits. Next, each client calculates its own 20 hash values (`H1`$\sim$ `H20`) with each participant's public key (i.e., A_pub and B_pub) and sets the corresponding bit to 1 for each hash value as the bit array index (①). The generated bloom filter is exchanged and validated by each client using a commitment scheme. For example, to identify whether client A signs in the transaction, all the clients calculate their own 20 hash values (`H1`$\sim$ `H20`) with client A's public key. They then check the corresponding bit for each hash value as the bit array index (②). If all values are 1, it indicates that client A has participated in the signing of the transaction. This process is performed for client B in the same manner. After verifying that the bloom filter indicates that all clients (clients A and B) have participated in the signing process, the clients store the bloom filter in the transaction. If the validation fails, the signing process is stopped.

After generating the bloom filter, the clients perform the process of signing the transaction. In our scheme, we distribute the authority of signing a transaction by using the T-ECDSA for high security. For this, we use multi-party computation (MPC) in the T-ECDSA which is a method for creating a single signature by distributing the authority to multiple clients. In this process, to

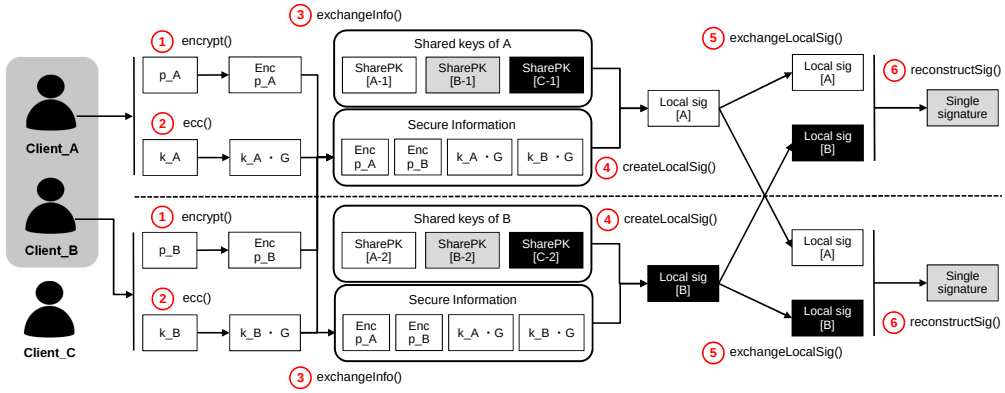

Figure 3.5: A signing process for a 2 of 3 multi-signature wallet (p: random number, k: random number, G: generate point, sig: signature)

distribute the authority and hide information, we generate random numbers and a local signature for each client, respectively. Figure 3.5 shows the signing process of a 2 of 3 multi-signature wallet. Clients A and B first generate their own random numbers (k and p), which are used to generate a single signature via the T-ECDSA. For example, clients A and B generate random numbers such as p_A/k_A and p_B/k_B, respectively. The parameter k is used to generate a local signature and p is used to hide its own shared keys in the signing process. Next, the clients encrypt their own p via `encrypt()` (①). Each client then calculates a point of an elliptic curve with its own random k and generation point G via `ecc()` (②).

To generate a local signature, the clients exchange the encrypted p (i.e., enc_p_A and enc_p_B) and the points (i.e., $k\_A \cdot G$ and $k\_B \cdot G$) with each other via `exchangeInfo()` (③). The clients load the generated information (shared keys, private keys, etc) mentioned in Section 3.2.3. Subsequently, the clients generate their local signatures ($local\_sig\_A$ and $local\_sig\_B$) with the shared keys and secure information (i.e., enc_p_A, enc_p_B, $k\_A \cdot G$, and $k\_B \cdot G$) via `createLocalSig()` (④). Each local signature is part of a single signature and

contains information about each participant's shared keys encrypted. The signatures are used to reconstruct a complete single signature. After creating the local signature, the clients exchange their local signature via `exchangeLocalSig()` (⑤). Furthermore, each client completes signing the transaction by reconstructing the local signatures to obtain a single signature via `reconstructSig()` (⑥). A signed transaction is generated for each participant, and the transaction is sent to the blockchain by client A who creates a channel in the preprocessing step of Section 3.2.2 by the wallet rule.

Unlike Shamir's secret sharing [69] scheme, the process of generating a single signature is more secure because in the process, each local signature, rather than a common private key, is reconstructed for signing the transaction. Therefore, each participant cannot acquire secure information such as private keys and shared keys, and a malicious client cannot sign another transaction with the information. In contrast to other multi-signature wallets [13, 14], the proposed wallet does not include any participant's public key or wallet address since it uses a bloom filter, thereby providing more privacy.

3.2.5 Identifying a participant of a transaction

After signing a transaction, the proposed multi-signature wallet identifies who signs the transaction in order to hold clients accountable for the transfer of digital assets. For this, the *validation module* in a client downloads the transaction data from blockchains for identifying a participant in a transaction. The module extracts the bloom filter from the transaction and checks who signs the transaction via the bloom filter. For example, as shown in Figure 3.4, the Bloom-filter identifies whether client A signs a transaction. If all the values of the index corresponding to the hash value of client A's public key are 1, we can identify that client A signed the transaction. Otherwise, we can identify

that client A does not sign the transaction. Through these processes, we can identify the participant via the bloom filter values, without exposing private information.

3.3 Evaluation

3.3.1 Experimental setup

In our evaluation, we use a server node comprising two Intel Xeon E5-2683 processors (total 32 cores), 64 GiB DRAM, and running Ubuntu 16.04.5 LTS. We use ten client nodes, and each client includes an Intel i7 processor and 32 GiB DRAM running macOS. We evaluate the existing multi-signature wallets go-bitcoin-multisig [70] and MultiSigWallet [13] and our proposed wallet on two blockchains (i.e., Bitcoin and Ethereum). We compare the proposed wallet with go-bitcoin-multisig supporting multi-signature and using pay-to-script-hash (P2SH) for the Bitcoin protocol. We also compare our multi-signature wallet with MultiSigWallet, which uses a smart contract, since the Ethereum blockchain does not support multi-signature as a protocol. We evaluate the transaction validation time, transaction size, and transaction fees (cost), and examine the privacy. Note that we measure the performance for 1 of 1, 2 of 2, and 2 of 3 since Bitcoin supports a multi-signature up to 2 of 3. In Ethereum, generally, the number of participants is ten, and therefore, we measure the performance in the case of 1 of 1, 2 of 2, 2 of 3, 3 of 4, 4 of 5, and 5 of 10.

3.3.2 Performance results

Transaction validation time. The left side of Figure 3.6 shows the transaction validation time of one of the existing wallets (go-bitcoin-multisig) and our proposed wallet on the Bitcoin blockchain. We compare multi-signature until 2 of 3 participants since Bitcoin supports multi-signature up to 2 of 3 by default.

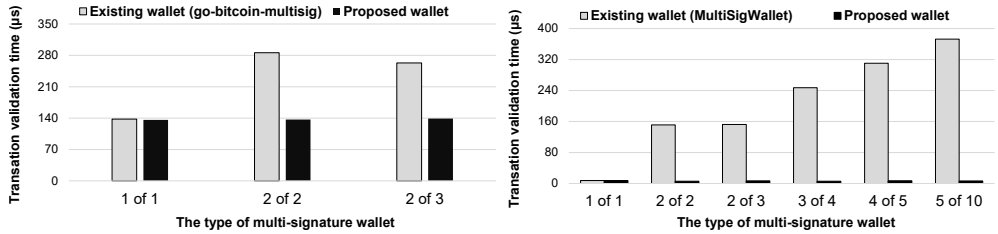

Figure 3.6: Transaction Validation Time in Two Networks: Bitcoin Blockchain (left) and Ethereum Blockchain (right)

In the case of 1 of 1, the performance as a baseline of the existing and proposed schemes is equal since both wallets use the general single signature scheme. For the multi-signature types 2 of 2 and 2 of 3, the proposed wallet shows performance improvement by up to 2x and 1.89x compared with the existing wallet, respectively. The transaction signed by the existing wallet requires more validation time since multiple signatures have to be validated. By contrast, since the proposed wallet signs a transaction with a single signature, the verification time is the same as that of a single-signature scheme. Therefore, the proposed wallet shows higher validation performance than the existing multi-signature wallet.

The right side of Figure 3.6 shows the transaction validation time for the other existing wallet (MultiSigWallet) and our proposed wallet on the Ethereum blockchain. Likewise, the performance of the wallet is equal for the existing and proposed schemes for 1 of 1. For the multi-signature types 2 of 2, 2 of 3, 3 of 4, 4 of 5, and 5 of 10, the proposed wallet shows performance improvements by up to 24.7x, 22.9x, 41.1x, 44.3x, and 58.3x compared with the existing wallet, respectively. MultiSigWallet generates multiple smart contract transactions such as submit and confirm transactions, and these transactions should be validated by executing the smart contract on Ethereum, which increases the validation time. Since it generates only a single transaction, the proposed wallet shows

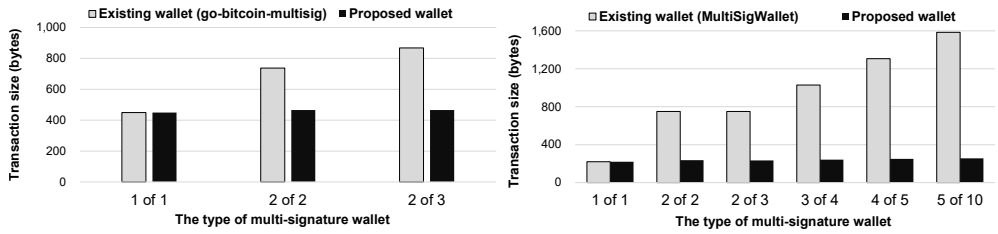

Figure 3.7: Transaction Size in Two Networks: Bitcoin Blockchain (left) and Ethereum Blockchain (right)

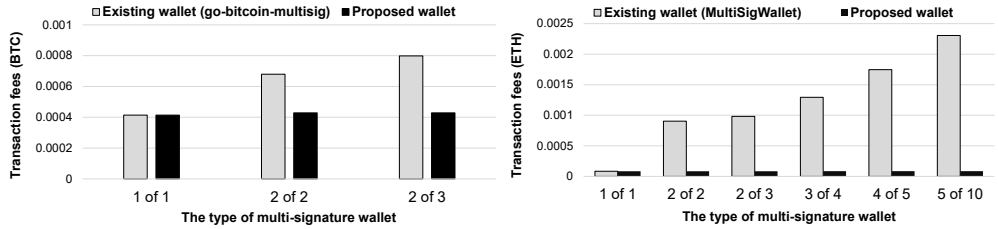

Figure 3.8: Transaction Fee in Two Networks: Bitcoin Blockchain (left) and Ethereum Blockchain (right)(BTC: bitcoin, ETH: ethereum)

higher validation performance than the existing wallet.

Transaction size. The left side of Figure 3.7 shows the transaction size of the existing wallet (go-bitcoin-multisig) and our proposed wallet on the Bitcoin blockchain. The proposed wallet reduces the transaction size by up to 1.58x and 1.86x compared with the existing wallet for the multi-signature types 2 of 2 and 2 of 3, respectively. This is because the existing wallet signs a transaction with multiple signatures. Moreover, a transaction of the existing wallet involves all participants' public keys for validating a transaction. Therefore, as the number of participants increases, the transaction size of the existing wallet increases. By contrast, the proposed wallet signs a transaction with a single signature, a bloom filter, and a common public key for validating a transaction, which reduces the transaction size.

The right side of Figure 3.7 shows the transaction size of the existing wallet

(MultiSigWallet) and our proposed wallet on the Ethereum blockchain. The proposed wallet reduces the transaction size by up to 3.15x, 3.2x, 4.24x, 5.2x, and 6.18x compared with the existing wallet for the multi-signature types 2 of 2, 2 of 3, 3 of 4, 4 of 5, and 5 of 10, respectively. As the number of participants increases, the total transaction size of the existing wallet increases since the number of multiple transactions increases. By contrast, in the case of the proposed wallet, the transaction size is almost constant or increases only slightly. The reason is that the wallet generates a single transaction even though the number of participants increases and the size of the bloom filter increases slightly according to the number of participants.

Transaction fee. A client should pay a transaction fee when transmitting a digital asset on a blockchain. In the case of Bitcoin, the transaction fee depends on the transaction confirmation time[4] Before the final transaction fee is determined, an optimal transaction fee per byte is decided by considering the Bitcoin network status and transaction confirmation time by using a Bitcoin fee calculator [71]. The final transaction fee is decided by the optimal transaction fee and the transaction size. The left side of Figure 3.8 shows the transaction fee of the existing wallet (go-bitcoin-multisig) and the proposed wallet on the Bitcoin blockchain. The proposed wallet reduces a transaction fee by up to 1.58x and 1.86x compared with the existing wallet for the multi-signature types 2 of 2 and 2 of 3, respectively. Thus, we could reduce the transaction fee by reducing the transaction size through the use of a single signature.

In the case of Ethereum, the transaction fee depends on gas usage and gas price[5]. Before determining the transaction fee, an optimal gas price is decided

[4]The transaction confirmation time is the time interval from the beginning and the approval of the transaction on a blockchain. We set this confirmation time as 60 minutes based on a previous study [56].

[5]The unit of transaction fee in Ethereum is gas

by considering the Ethereum network status and the transaction confirmation time via an Ethereum gas station [72]. The transaction fee is then determined by the optimal gas price and gas usage. The gas usage is charged cumulatively whenever a smart contract is executed or extra data is included. The right side of Figure 3.8 shows the transaction fees of the existing wallet (MultiSigWallet) and our proposed wallet on the Ethereum blockchain. The proposed wallet reduces the transaction fee by up to 10.7x, 11.6x, 15.2x, 20.7x, and 27x compared with the existing wallet for the multi-signature types 2 of 2, 2 of 3, 3 of 4, 4 of 5, and 5 of 10, respectively. The existing wallet creates multiple transactions and executes a smart contract to sign a transaction, which incurs more gas usage. By contrast, our scheme does not require the execution of a smart contract. Although additional data is obtained from the bloom filter, it hardly affects gas usage. Consequently, the proposed wallet incurs a lower transaction fee than the existing wallet.

3.3.3 Discussion of usecase

Multi-signature wallets bring significant security enhancements to the blockchain ecosystem. These wallets require multiple keys to authorize a transaction, providing a more secure environment compared to conventional wallets.

One of the primary areas of application for multi-signature wallets is in the management of corporate funds within an organization. In a traditional setting, the handling of an organization's funds is typically done through a centralized system with checks and balances in place. For instance, a transaction might require the signatures of several key members of the organization. Implementing multi-signature wallets can replicate this secure environment in a decentralized manner. Transactions can be set to require the authorization of multiple key holders, ensuring that no single person can misappropriate funds.

Additionally, multi-signature wallets are useful in managing shared funds in less formal situations, such as family trusts or joint accounts. In such cases, the requirement for multiple signatures can serve as protection against one party misusing the funds, making the management of shared assets more secure and transparent.

Finally, multi-signature wallets can also serve as a security mechanism for individual users. By distributing the keys across multiple devices or locations, users can protect themselves against the loss or theft of a single key. This security measure is particularly relevant in situations where large amounts of assets are being stored, making the potential losses from a single point of failure extremely high.

While the advantages of multi-signature wallets are clear, there are also challenges to be addressed. For instance, the need to manage multiple keys introduces additional complexity for users. Moreover, the technology underlying multi-signature wallets is still evolving, and there may be unforeseen technical challenges to overcome. Therefore, further research and development are needed to fully realize the potential of multi-signature wallets.

3.3.4 Discussion of privacy

Any client can access data on public blockchains such as Bitcoin and Ethereum. Thus, there can be privacy problems for the existing wallets on public blockchains since the blockchains can store participants' information such as public key and wallet address. For example, in the case of MultiSigWallet, a signed transaction on the blockchain includes the associated participants' wallet addresses. In the case of go-bitcoin-multisig, a signed transaction on the blockchain includes all participants' public keys that are required to verify the transaction. Consequently, in both existing schemes, there can be privacy problems since the

participants' information (public keys or wallet addresses) on the blockchain can be exposed to any client. In the proposed wallet, the signed transaction includes a common public key and a bloom filter instead of the participants' information. Therefore, the proposed wallet can provide more privacy than the existing wallets by preventing the exposure of participants' information.

3.4 Summary

We investigate blockchain wallets that support a digital signature in blockchains and found that while a multi-signature wallet provides higher security than a single-signature wallet, it increases the transaction validation time and transaction size. Furthermore, the use of a multi-signature wallet requires the modification of the blockchain protocol. To overcome these issues, we develop a new multi-signature wallet with high performance and a small transaction size by using a T-ECDSA and a bloom filter. Through the bloom filter, users can reduce transaction size and preserve privacy. We implement the proposed multi-signature wallet using a Multi-party ECDSA and evaluate the wallet's performance for Bitcoin and Ethereum. Our experimental results show that the proposed multi-signature wallet reduces the validation time and transaction size more than other existing multi-signature wallets.

Chapter 4

Enabling SQL-Query Processing in Blockchain Systems

4.0.1 Motivation

By using smart contracts, users can implement a wide range of business logic such as distributed applications (DApp) on the blockchain.

Smart contracts frequently require the ability to obtain data in a range (such as purchase history and sales history). For example, Opensea, famous for its NFT market, shows that 121 million people visit the site every month [73]. However, as shown in Figure 4.1, the transactions are only 974,220 per month. In other words, it can be seen that the frequency of checking the details in each wallet is greater than the transaction frequency. However, a key-value store in the blockchain is challenging since it does not provide range query processing. In this instance, a smart contract can often retrieve range data in one of the following two ways:

- **Using user-defined data structure.** As one method for retrieving the

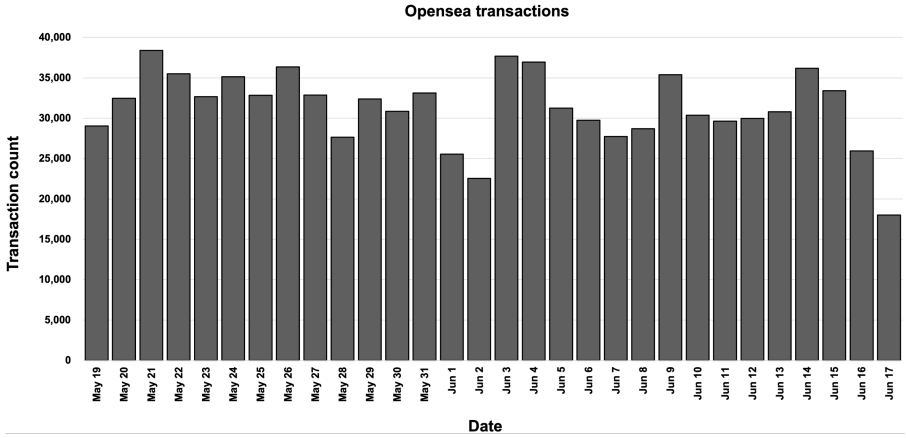

Figure 4.1: Opensea transaction statistics May-June 2023

range and condition data of a smart contract, a user-defined data structure can be used in the smart contract. For example, when a user requests the states in a range, the blockchain system retrieves the requested states, saves the results in the data structure (e.g., an array or map), and gives it to the user. Even though this method can provide a range of data, it can decrease the overall performance. The reason is that an Ethereum Virtual Machine (EVM) is loaded and it runs the smart contract by reading states from the database one by one instead of a range query.

- **Using an external relational database.** To retrieve the range and condition data of a smart contract and regular transaction, we can use an external database in a blockchain system. For serving the transaction history, usually, DApp collects all transactional data occurring on the blockchain is gathered, with a specific focus on transactions taking place through the DApp smart contract. After that, the Collected transaction's event history is stored in an external database. Finally, when a user queries a transaction on the DApp, the DApp then retrieves the relevant

data from the external database and provides the results to the user. The external database enables the range query processing and so it provides higher search performance. However, this method can require additional management in the blockchain system. For instance, we should manually carry out many tasks when a blockchain node connects to an external database system (e.g., setting database APIs, constructing a database, and making tables). However, in our proposal, each node in a blockchain system has an embedded database system to automatically carry out these processes. By doing this, the user may quickly connect to the database system and leverage the embedded database system's range query processing.

There are multiple challenges associated with providing services that rely on data from the blockchain to external databases. Firstly, it is critically important for the service provider to maintain data consistency between the blockchain and the external database. Each new transaction occurring on the blockchain requires immediate updates to the external database, which can introduce significant technical complexities. Secondly, the use of external databases, which are inherently centralized systems, introduces significant security concerns. It means that there is a possibility that Oracle problems [74] may occur in the situation in which data is transferred. Lastly, the fundamental design of blockchain technology promotes data that is public and accessible to all. However, this principle could be compromised when using an external database. In such a case, the company operating the database assumes full ownership, which could limit data accessibility. This poses a potential contradiction to the inherent decentralization ethos of blockchain technology. To solve this problem, we would like to introduce a method of exploring the range of data through SQL within

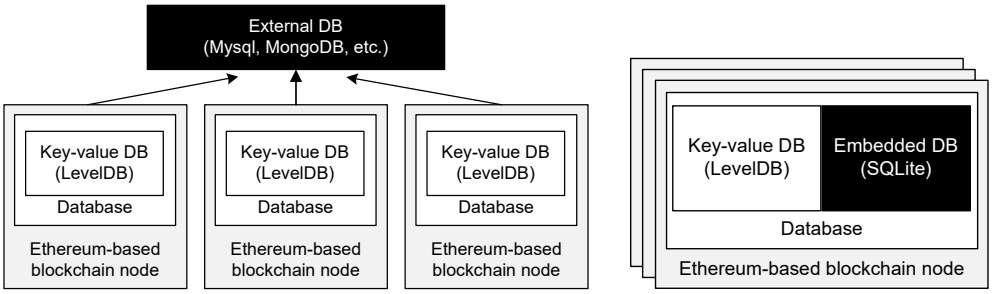

Figure 4.2: Classification of the database on Ethereum-based blockchain node (Left: external database, Right: Embedded database)

the blockchain.

4.1 Design and Implementation

To achieve higher searching performance and reduce the management cost in a blockchain system, we aim to enable the retrieval of range data and conditional data in the blockchain system. To do this, we combine an embedded database system into the blockchain system that provides SQL query operations for a range of data inside a blockchain without manually building an external database and a user-defined data structure.

Figure 4.2 shows the difference between an external database system and an embedded database system. It is whether the database is built in an application or not [75]. As shown in left Figure 4.2, an external database has to install a standalone application and centralized server. On the other hand, as shown in right Figure 4.2, the embedded database is built into an application and stores data among users without installing a separate database server [76]. Also, when using the external database for retrieving the information of the regular transaction, the blockchain system has to rely on a third entity (e.g., a DBMS server). Meanwhile, when using the embedded database, the blockchain system does not need to rely on a third entity.

Due to these embedded database advantages, we select the embedded databa se as the database in the blockchain node. Therefore, through the embedded database, we can provide a decentralized architecture and enable SQL query operations for retrieving range and conditional data. For an embedded database, we choose SQLite which is an embedded relational database system. SQLite is an open-source Database Management System (DBMS) and a lightweight embedded database that manages data with a single file and does not require a separate database server. So, via SQLite, we can be built into a client application [77].

Meanwhile, we maintain an existing key-value database (LevelDB) used in the blockchain system to exploit its advantages. For example, when a key-value database retrieves a block or a transaction to validate the transaction or block, it provides better performance in searching. Also, our system does not sacrifice the consistency of existing blockchain systems.

4.1.1 Design

Figure 4.3 shows the system overview(architecture) of the existing and proposed systems. In the existing system, as shown in the left side of Figure 4.3, when the service interface layer is transmitted from the application layer to an Ethereum-based blockchain system, it gets the transaction from the application layer. Then, it is sent from the service interface layer to the transaction layer. After then, the transaction layer classifies if the transaction type is a smart contract transaction or a regular transaction. If the type of transaction is a smart contract, the smart contract manager performs the smart contract transaction on Ethereum Virtual Machine(EVM). Then, the smart contract manager validates the performed transaction result. And, if the result is valid, the manager stores the transaction to `mempool` as a pending transaction.

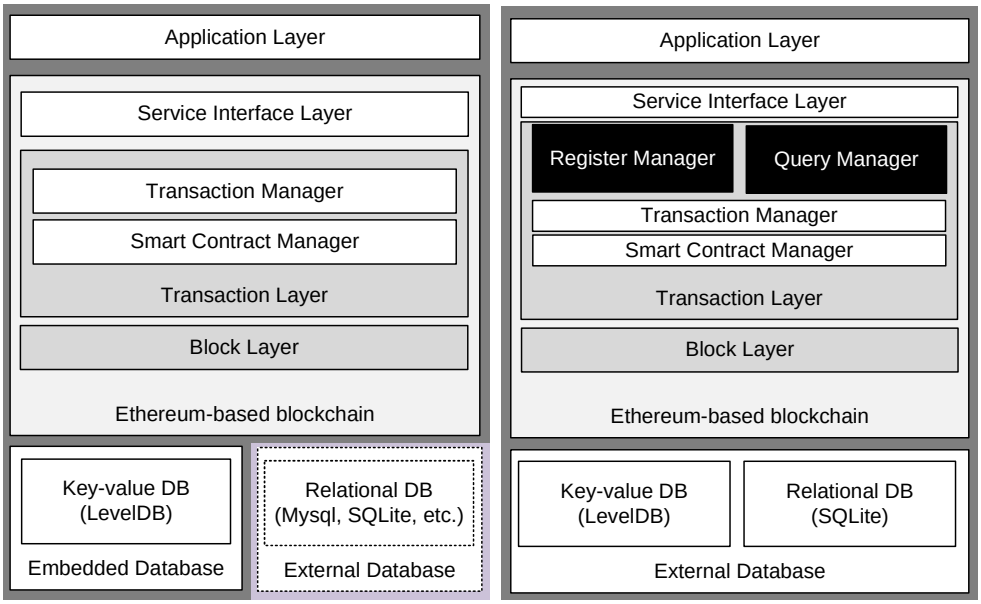

Figure 4.3: System overview (left: existing system, right: proposed system)

On the other hand, if the type of transaction is regular, the transaction manager verifies the regular transaction by checking the sender's balance in the transaction. After the transaction is verified, the transaction manager stores the transaction as a pending transaction in a transaction pool (`mempool`) which has the validated pending transactions.

When a block has to be generated, the block layer generates a block with pending transactions. The pending transactions include regular and smart contract transactions in the `mempool`. At this time, the pending transactions are performed so that the states are updated according to the transaction results. Finally, the information of the block with transactions is stored in a key-value database (i.e., LevelDB). Because the blockchain constructs a database as a key-value database, it is impossible to search for conditions or ranges in the existing blockchain. Therefore, a separate database has been established for retrieving the range and condition of blockchain data (e.g., smart contracts and

regular transactions).

Meanwhile, as shown in the right side of Figure 4.3, in the proposed system, we add a register manager and a query manager and modify the block layer. The managers handle smart contract transactions and regular transactions. The register manager registers the smart contracts and wallet addresses for regular transactions which are requested to retrieve range data through application and service interface layers. Then, like the existing system, the transaction manager and the smart contract manager perform the transactions, validates the transaction results, and store the transactions to `mempool`. Also, when a block should be generated, the block layer generates a block with pending transactions of which type is both smart contracts and regular transactions in `mempool`. Meanwhile, our modified block layer checks whether the transaction is associated with a smart contract or wallet address already registered by the register manager. If the transaction is associated with the smart contract or wallet address, the block layer stores the transaction in an embedded relational database (e.g., SQLite). After then, when a user requests the range and condition data of the smart contract and regular transaction, the query manager performs various SQL queries by using the database according to the requested operations.

Register Manager

For more efficiency, we register the smart contracts and wallet addresses only requested by users for retrieving the range and condition data to avoid managing all the transactions. This strategy can identify whether a transaction is stored or not in the embedded relational database (i.e., SQLite) according to the registered smart contract and wallet address.

To do this, we devise the register manager as shown in the bottom side of Figure 4.3. In terms of smart contracts, we create APIs in the register manager

such as `registerContractAddress` which registers the smart contract. The register manager stores the smart contract address which is used to identify the smart contract in SQLite when it gets the request to register a smart contract with `registerContractAddress`. Also, in terms of regular transactions, we create APIs in the register manager such as `registerWalletAddress` which registers the wallet address. When the register manager gets the request to register the wallet address via `registerWalletAddress`, it stores the wallet address in SQLite.

After the registration, the results of transactions from the registered smart contract and the wallet address are stored in SQLite by the block layer as shown in right Figure 4.3. Through registration, we can retrieve and track the results of transactions in a range and conditions. We will explain this mechanism for storing the results of smart contracts and regular transactions in SQLite in Section 4.1.1. Meanwhile, if the results of the smart contract and user do not need anymore, we can remove the corresponding smart contract and wallet address via `removeContractAddress` and `removeWalletAddress`. After then, the tracking for the smart contract and wallet address is stopped.

Query Manager

Query manager performs a SQL query for retrieving data of smart contracts and regular transactions by using the embedded relational database system (i.e., SQLite). To do this, we create an API such as `getData` for retrieving data in the smart contract. When the query manager has received the request for retrieval via `getData`, the query manager retrieves the data of the smart contract and regular transaction from SQLite. Also, in the query manager, only the `SELECT` query is executed, meanwhile, other queries (i.e., `INSERT`, `UPDATE`, and `DELETE`) are filtered to prevent modifying the data from outside. With SQLite,

the query manager can perform range or conditional query operations by calling `getData`. If a smart contract or wallet address is not registered through the register manager, the query manager does not perform any operations for the corresponding smart contract or regular transaction.

Block Layer

In our system, the modified block layer stores a block that includes smart contract transactions and regular transactions to both the key-value database (i.e., LevelDB) and relational database (i.e., SQLite). The block layer performs a two-level check operation. First, the block layer checks whether each transaction in a block is a smart contract transaction or a regular transaction. Second, if the transaction is a smart contract transaction, the block layer checks whether the smart contract transaction is associated with the smart contract already registered by the register manager.

Also, if the transaction is a regular transaction, the block layer checks the regular transaction is associated with the wallet address already registered by the register manager. If the transaction is registered via the register manager, the block layer stores the transaction in both LevelDB and SQLite. Otherwise, the transaction data is stored in LevelDB without storing the data in SQLite. It is because the key-value database is used for maintaining the Ethereum-based blockchain functionality for integrity, and the embedded relational database is used for retrieving range data in the smart contract. When the block layer receives the remove request via `removeContractAddress` or `removeWalletAddress` from the register manager, the block layer removes all the data related to the smart contract and wallet address in SQLite.

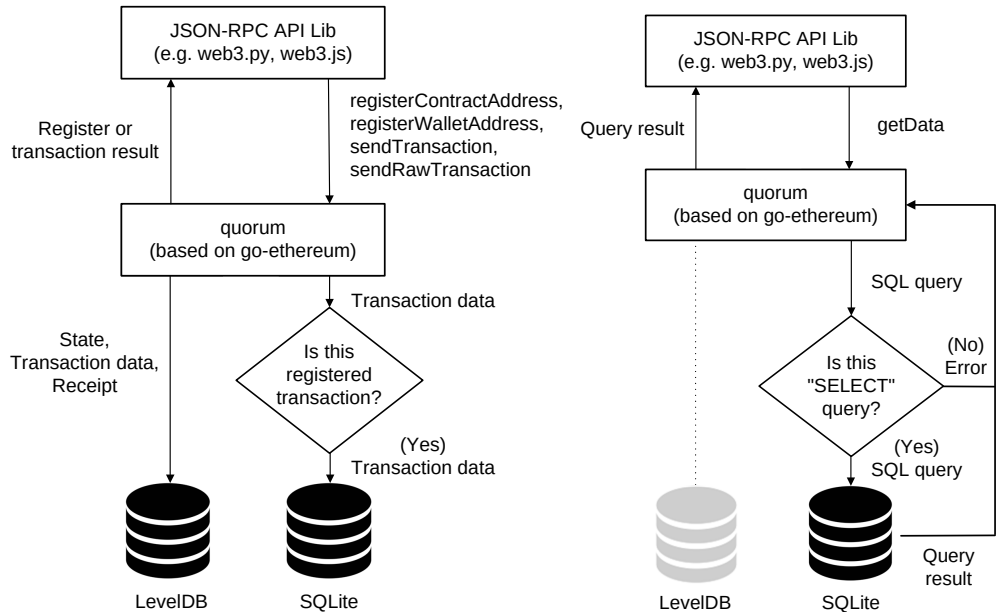

Figure 4.4: Process overview (left: register and store process, right: query process)

4.1.2 Implementation

We implement our scheme on *quorum* as shown in Figure 4.4. *quorum* is an Ethereum-based distributed ledger protocol with transaction/contract privacy and new consensus mechanisms (e.g., raft and Istanbul BFT) for a private blockc

hain. Also, we use the web3.py library of Python and SQLite3 library of `golang` for applications. SQLite3 is a driver conforming to the built-in *database/sql* interface in `golang`. web3.py and web3.js are a collection of libraries that allows interaction with a local or remote Ethereum-based blockchain node by calling JSON-RPC and using an HTTP or IPC (Inter-Process Communication) connection.

As shown in left Figure 4.4, to retrieve range and condition data in a smart

contract and regular transaction, a user requests the registration of an address of a smart contract via `registerContractAddress`. When the register manager receives the request, the register manager creates a table according to the smart contract in SQLite to store the transaction results. After the table is created successfully, the register manager stores the smart contract address in SQLite. In addition, to retrieve range and condition data in a regular transaction, a user requests the registration of a wallet address via `registerWalletAddress`. At that time, unlike a smart contract, the register manager stores the wallet address at the pre-defined table in SQLite without creating a table.

After the registration of a smart contract, the smart contract and regular transactions called by `sendTransaction` and `sendRawTransaction` from a user is received and processed via *quorum*. When the block layer stores a block with the transactions, the block layer checks whether each transaction in a block should be stored in SQLite for range query according to their registration via `checkIsTrackedContract`. If the block layer should store transaction data in SQLite, the block layer stores the transaction data in both SQLite and LevelDB. Otherwise, the block layer only stores the transaction data in LevelDB. For example, in the case of a smart contract transaction, the block layer stores the transaction data of the smart contract performed through EVM in SQLite via `insertValue(smart_contract_address, params)`. In the case of regular transactions, the block layer stores the regular transaction data in SQLite via `transfer(source address, destination address, amount)`.

The right side of Figure 4.4 shows the query process in our system. When range data in a smart contract needs to retrieve, JSON-RPC such as `getData(SQL_select_query)` is called. Then, as shown in the figure, only the `SELECT` query is performed in SQLite, and other SQL queries such as `INSERT`, `UPDATE`, and `DELETE` are eliminated via a regular expression. If the SQL query syntax is wrong

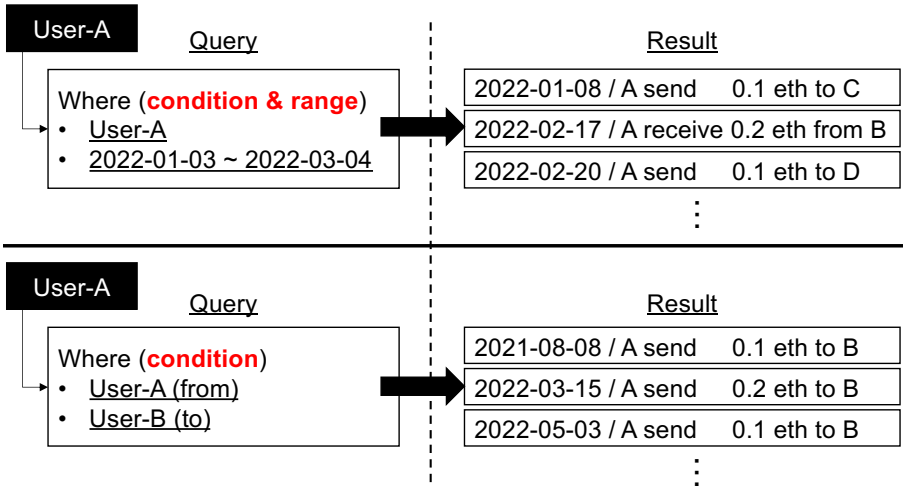

Figure 4.5: Retrieval regular transaction in the blockchain

or other queries, the query manager returns the error message. Otherwise, the query manager returns the result data of the syntax.

4.1.3 Usage

Figure 4.5 shows how to retrieve regular transactions in the blockchain. To retrieve the information of regular transactions for a certain period, a user requests to query with the user A's wallet address and the period. After then, the blockchain returns the result of the query request. The result is transaction information related to user A in the period. For example, user A sends 0.1 eths to user C, user A receives 0.2 eths from user B, and user A sends 0.1 eths to user D. In addition, user A can retrieve the transaction with a specific user. To do this, user A enters their address and another user's (user B) address. After then, the blockchain returns the result of the query request. The result is transaction information related to user A and user B. For example, user A sends 0.1 eth to user B, user A sends 0.2 eth to user B, and user A sends 0.1 eth to user B. In this case, the query manager checks the query types whether

it is related to regular transactions. After then, the query manager looks up the regular transaction table in SQLite, checks the search conditions, and responds to the query.

On the other hand, in the case of an existing system with an external database, it is necessary to build a separate database by synchronizing data information from the blockchain. The reason is that the database used in the blockchain is a key-value store, therefore, it is hard to retrieve conditions or ranges of data. So, it receives data from the blockchain at the time the block is generated and stores the data in a separate database such as MySQL. Afterward, the method of retrieving general transactions is similar to the proposed system, requesting a query to a separate service using an external database and receiving the result accordingly.

4.2 Evaluation

4.2.1 Experimental setup

We perform all of the experiments on five 32-core machines. Each has two Intel Xeon E5-2683 processors (without hyperthreading), 64 GiB DRAM, and runs Ubuntu 16.04.5 LTS distribution with Linux kernel 4.4.0. We use *golang* 1.10.7, *python* 3.7 and *jmeter* [78] which are used to evaluate applications. We empirically evaluate our proposed system by using a synthetic benchmark. In the case of a smart contract, the smart contract scenario of the synthetic benchmark is an energy usage storage application which a user stores electric energy usage every 15 minutes, and the total duration of storing the data is one year.

We make a smart contract for our evaluation. It consists of a variable and an array of user-defined data structures per user. The variable stores a timestamp that records the last time updated by a user. The array stores the actual energy

usage. The range to be retrieved in the smart contract is calculated as follows:

$$\begin{aligned} startIndex &= MNE - (LSTS - STS)/c - 1 \\ endIndex &= MNE - (LSTS - ETS)/c - 1 \\ &return\ array[startIndex : endIndex] \end{aligned} \tag{4.1}$$

`MNE` denotes the maximum number of entities in the smart contract during one year. In our evaluation, the number is 35,040. `STS` and `ETS` denote the start timestamp and end timestamp given by a user for retrieving the range data, respectively. `c` is a constant that represents seconds of the storing cycle. We set `c` as 900 seconds to convert 15 minutes to seconds. Using this smart contract, we evaluate the existing and proposed systems in terms of `INSERT` and `SELECT` performance.

In the case of a regular transaction, the regular transaction scenario of the benchmark is sending and receiving cryptocurrency between users. As it is a basic function of the blockchain, we exploit the internal function without writing a separate contract. In addition, to compare the existing system with an external database, we build the blockchain explorer which stores synchronized data of blocks and transactions from the blockchain with the external database. As the external database, we use MySQL since MySQL is typically used as an external database in the blockchain. We run each experiment with 10 measurements and report the average.

4.2.2 Performance results

SELECT performance

The top side of Figure 4.6 presents the `SELECT` performance of existing and proposed systems in the case of smart contracts. For experimental parameters, we set the number of threads as 1 and the number of entities as 10,000, 20,000,

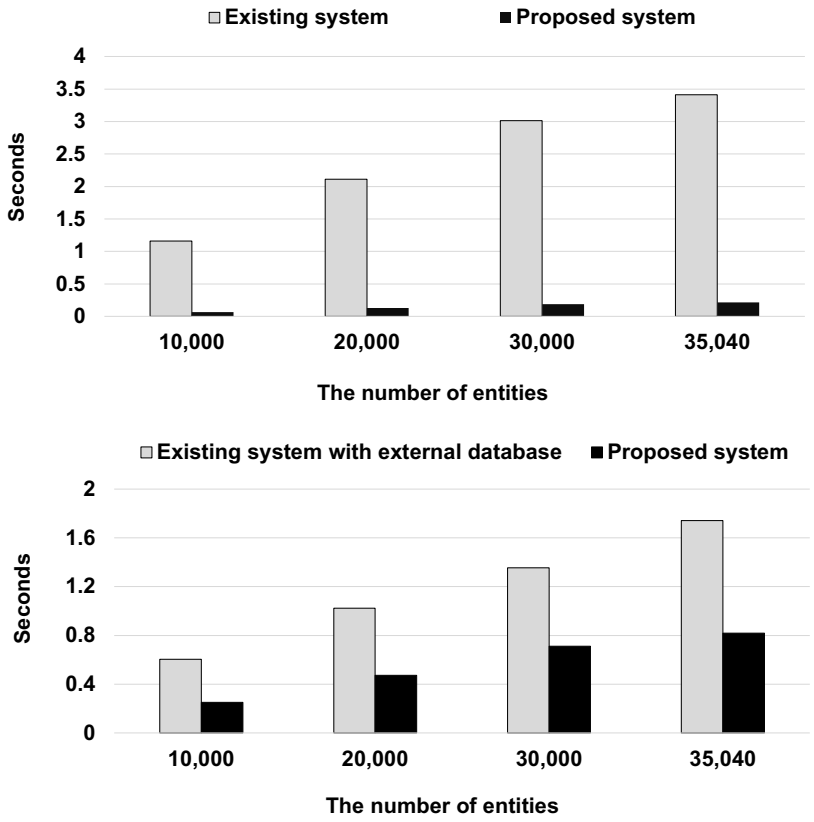

Figure 4.6: Execution time of select operations (top: smart contract, bottom: regular transaction)

30,000, and 35,040. Thus, it shows the performance results according to the number of entities. As shown in the figure, the proposed system improves the performance by up to 16.9x, 16.5x, 15.8x, and 15.7x compared with the existing system when the number of entities is 10,000, 20,000, 30,000, and 35,040, respectively. The proposed system provides a range query operation, while the existing system has to retrieve each data one by one without the range query. Therefore, it shows a better performance than the existing system. The execution time of the existing and proposed systems increases as the number of entities increases. Because, as the number of entities increases, the time of data

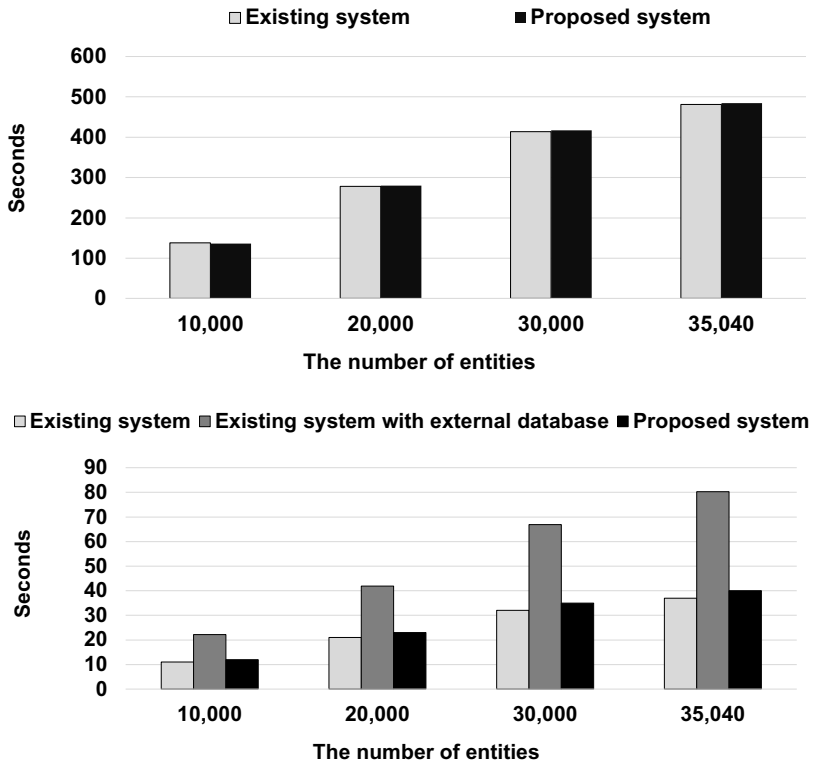

Figure 4.7: Throughput of insert operations (top: smart contract, bottom: regular transaction)

retrieval and the amount of data is increased. Also, this result shows that the execution time of the existing system increases rapidly, while the execution time of the proposed system increases slowly as the number of entities increases.

The bottom side of Figure 4.6 presents the `SELECT` performance of the existing system with the external database and proposed system in the case of regular transactions. For experimental parameters, like a smart contract experiment, we set the number of threads as 1 and the number of entities as 10,000, 20,000, 30,000, and 35,040. As shown in the figure, the proposed system improves the performance by up to 2.40x, 2.16x, 1.90x, and 2.12x compared with the existing system with an external database when the number of entities

is 10,000, 20,000, 30,000, and 35,040, respectively. It is because our proposed scheme combines SQLite for a regular transaction which is faster and simpler compared with the existing system with MySQL.

INSERT performance

The top side of Figure 4.7 presents the `INSERT` performance of existing and proposed systems. The experimental parameters used in the `INSERT` evaluation are the same as those used in the `SELECT` evaluation. The execution time of the proposed system increases by up to 1.013x, 0.994x, 0.992x, and 0.993x compared with the existing system when the number of entities is 10,000, 20,000, 30,000, and 35,040, respectively. This result shows a minor overhead. In terms of throughput, the proposed system provides 73.3, 71.4, 71.9, and 72.3 entities/s and the existing system provides 72.3, 71.8, 72.4, and 72.7 entities/s when the number of entities is 10,000, 20,000, 30,000, and 35,040, respectively. These results present the throughput of `INSERT` operations of the proposed system as almost the same as that of the existing system although we support additional embedded relational databases (i.e., SQLite) for fast retrieval.

The bottom side of Figure 4.7 presents the `INSERT` performance of an existing system with an external database and proposed system. The experimental parameters used in the `INSERT` evaluation are the same as those used in the `SELECT` evaluation. The proposed system increases the execution time by up to 1.090x, 1.095x, 1.093x, and 1.081x compared with the existing system when the number of entities is 10,000, 20,000, 30,000, and 35,040, respectively. It is because the proposed system stores additional data at the SQLite in the blockchain. Meanwhile, the proposed system decreases the execution time by up to 1.84x, 1.82x, 1.91x, and 2.00x compared with the existing system with an external database when the number of entities is 10,000, 20,000,

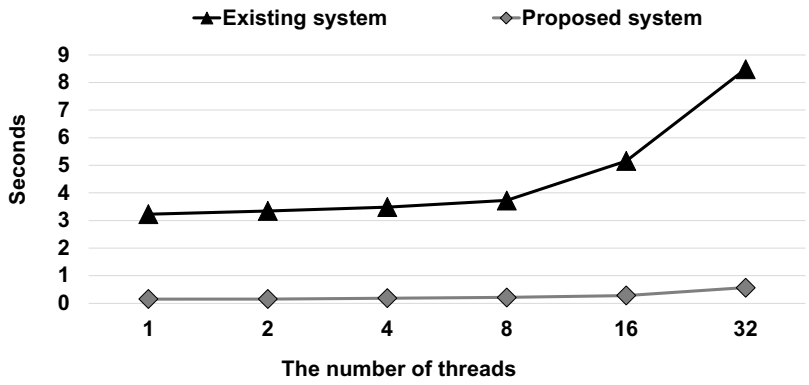

Figure 4.8: Execution time of select operations in smart contract

30,000, and 35,040, respectively. It is because the existing system with external database stores additional data at the external database outside the blockchain. The external database requires additional synchronizing operations with the blockchain. However, the proposed system stores the data in the embedded database (SQLite) inside the blockchain without synchronizing operations. Therefore, the result demonstrates that the proposed scheme shows a better performance than the existing system with an external database.

Note that all the experimental results by regular transactions in the bottom side of Figure 4.7 show better performance than those by the smart contract of the top side of Figure 4.7. Because the smart contract transaction should be executed by the smart contract function by the Ethereum Virtual Machine (EVM), it takes a longer time than a regular transaction.

4.2.3 Impact on the number of threads

Figure 4.8 and Figure 4.9 present the performance in the case of `SELECT` operations with the different numbers of threads. As shown in the figure, in all systems, the execution time increases as the number of threads increases. In terms of smart contract, the proposed system improves the performance by up

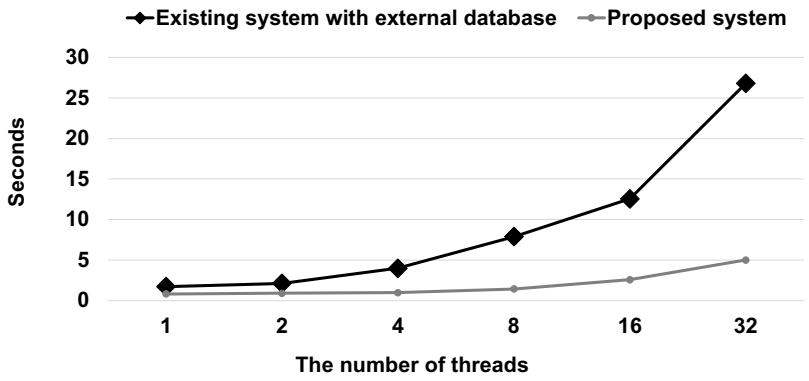

Figure 4.9: Execution time of insert operations in regular transaction

to 21.1x, 22x, 18.7x, 17.4x, 18.4x, and 15x compared with the existing system when the number of threads is 1, 2, 4, 8, 16, and 32, respectively. Especially, in the existing system, the execution time increases rapidly when the number of threads is beyond 16 threads. The execution time of the existing system is higher by up to about 8 seconds compared with that of the proposed system.

In terms of the regular transaction, the proposed system improves the performance by up to 2.12x, 2.31x, 4.02x, 5.46x, 4.87x, and 5.34x compared with the existing system with an external database when the number of threads is 1, 2, 4, 8, 16, and 32, respectively. Especially, in the case of the existing system with an external database, the execution time increases rapidly when the number of threads is beyond 16 threads. The execution time of the existing system with the external database is higher by up to about 21 seconds compared with that of the proposed system.

4.2.4 Measuring resource usage

To measure resource usage for one node when performing `SELECT` operations, we set the number of entities as 35,040 (one-year) and measure the resource usage from the program start to the termination. Also, we set the 35,040 regular

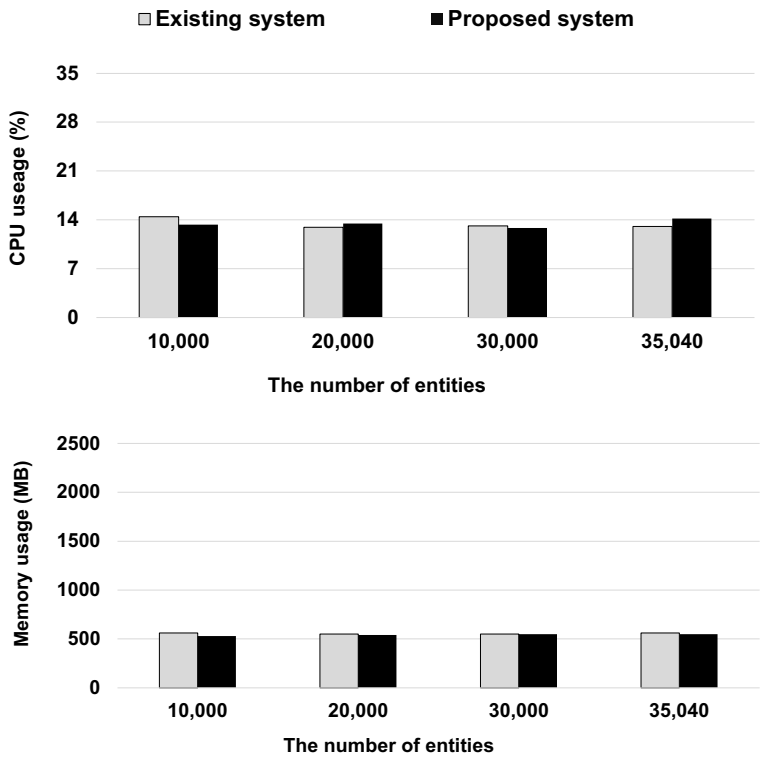

Figure 4.10: Resource usage with the different number of entities (top: CPU, bottom: memory)

transactions the same as the smart contract. Figure 4.13 shows the CPU and memory usage according to the number of threads and the number of entities in the smart contract and regular transaction. As shown in Figure 4.12, the CPU and memory usage is almost the same when the number of entities increases and the number of threads is only one. It is because this number of entities used in our evaluation does not significantly affect memory usage.

Meanwhile, as shown in Figures 4.13, CPU and memory usage increases as the number of threads increases. In this evaluation, the multiple threads process the entities in parallel. Thus, the required resource increases at the same time. We note that as the number of threads increases, the required amount

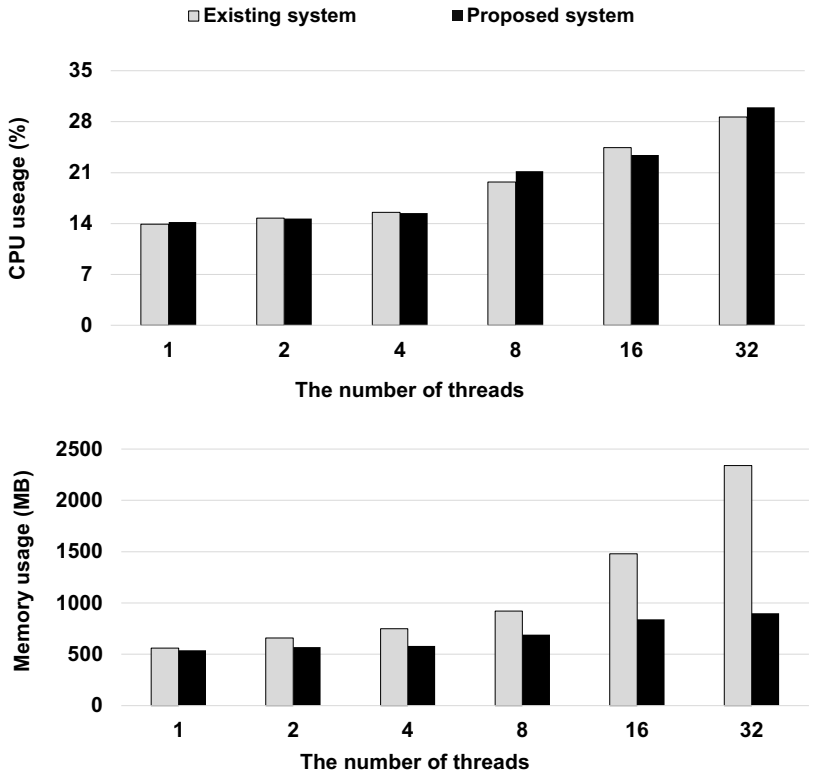

Figure 4.11: Resource usage with the different number of threads (left: CPU, right: memory)

of memory in the existing system increases by up to 2.6x compared with the proposed system. The results show that, in the proposed system, EVM uses more memory than SQLite does. The reason is that more EVMs are required to support user-defined data structures when more threads are added to the existing system.

In the regular transaction, as shown in Figure 4.12, like smart contract, the CPU usage in the proposed system is higher than that of the existing system with an external database when the number of entities increases. Even if the CPU usage in the proposed system is higher, the CPU usage of the proposed system is average of about 2% and it shows that the CPU usage itself is still low.

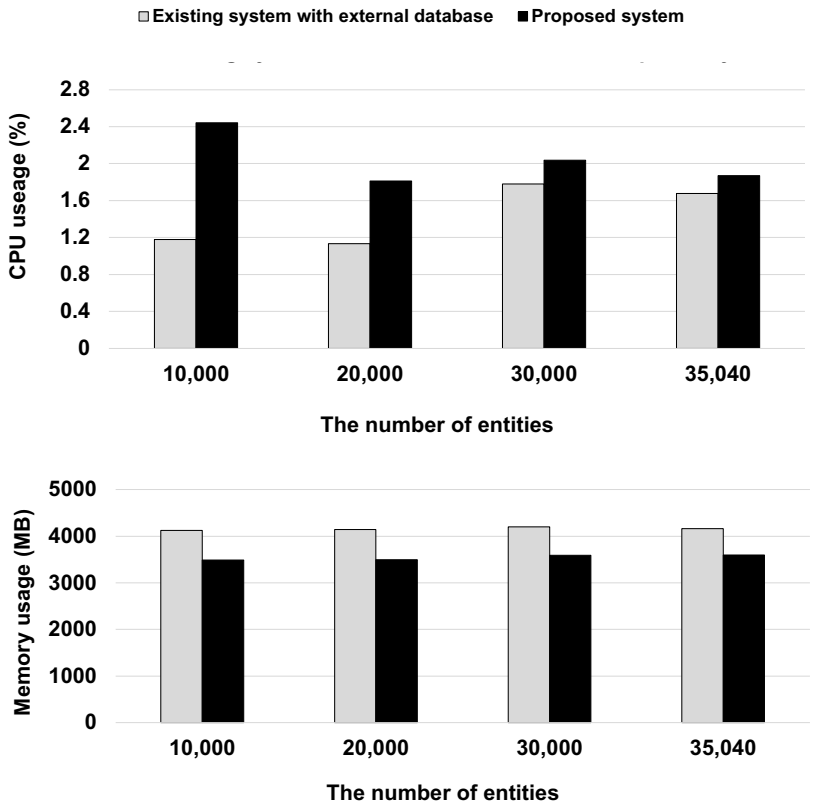

Figure 4.12: Resource usage with the different number of entities (left: CPU, right: memory)

Also, the memory usage in both systems is similar even if the number of entities increases. The top side of Figure 4.13 shows that the CPU usage increases as the number of threads increases in the proposed system. Meanwhile, the existing system with an external database does not increase CPU usage. Since the existing system with an external database is a more complex architecture (e.g., locking mechanism) than SQLite, the CPU usage is almost the same even if the number of threads increases. Finally, the bottom side of Figure 4.13 shows that both systems slightly increase memory usage according to the number of threads.

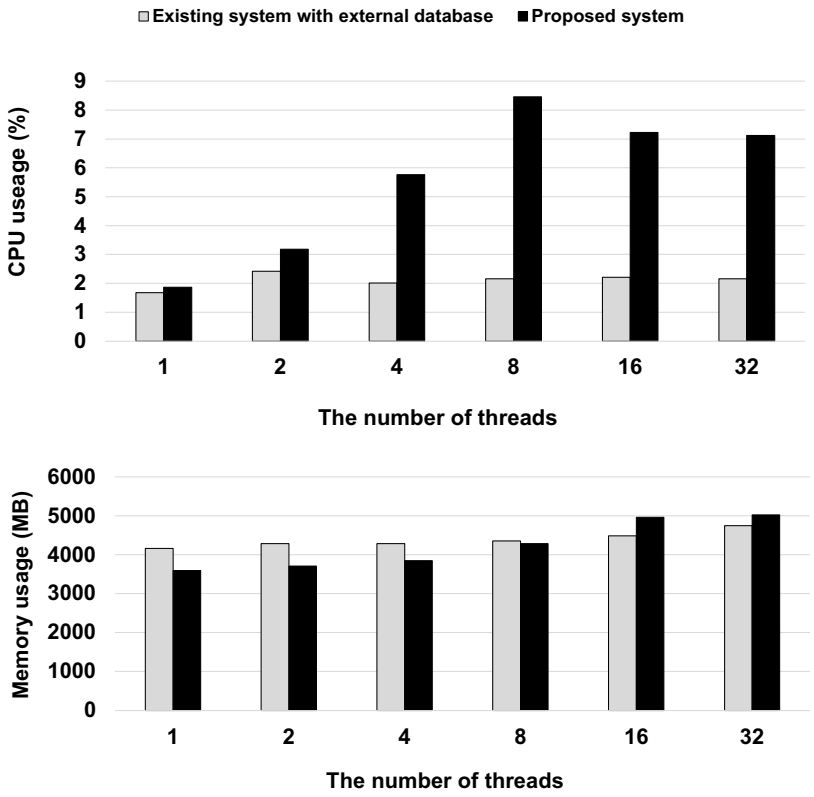

Figure 4.13: Resource usage with the different number of entities (left: CPU, right: memory)

4.2.5 Byzantine Fault Tolerant

The left side of Figure 4.14 shows the existing system using an external database. When an existing system external database is used, the external database is a centralized system and can be attacked by security. Therefore, if the database is attacked or corrupted, users may not be able to view transaction data properly. On the other hand, as shown right side of Figure 4.14, if the proposed system is used using the embedded database, users can access data on multiple blockchains and check the correct data. In addition, the blockchain provides public data that anyone can access in principle. However, while the company

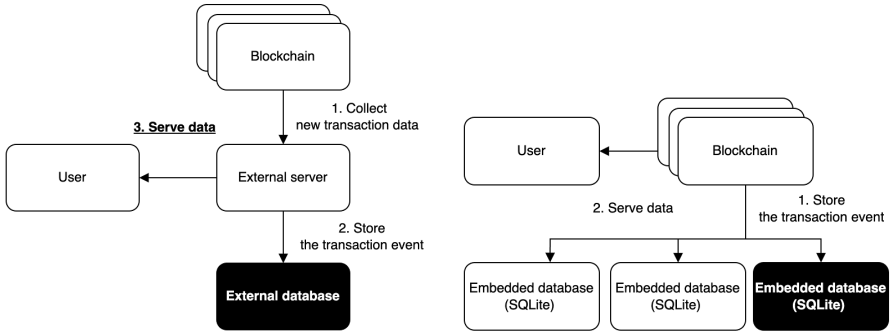

Figure 4.14: Influence on Byzantine Fault Tolerant according to use in a database (left: external database, right: embedded database)

operating the external database has full ownership of the database, the proposed system allows each blockchain node to take full ownership of it. This is a way to uphold blockchain principles.

4.3 Conclusions

In this article, we enable SQL query operations in a blockchain system. To do this, we combine an embedded relational database with an Ethereum-based blockchain system to provide SQL queries. This enables range query for the smart contract without any user-defined data structure and decreases the management cost for the regular transaction without any external database. We have implemented the proposed scheme on an Ethereum-based blockchain system and evaluated the proposed system using a synthetic benchmark. Our experiment results show that the proposed system in the case of smart contracts can improve the performance by up to about 22x compared with the existing system. Also, our system shows a similar search performance compared with the existing system including an external database in the case of regular transactions.

Chapter 5

Proof of Double Committee for Decentralization Consensus Algorithm in Blockchain system

5.1 Motivation

5.1.1 Centralization of Blockchain

The issue of centralization in blockchain technology, notably in Ethereum's transition to a Proof-of-Stake (PoS) mechanism, has become a subject of significant concern within the cryptocurrency industry. These concerns predominantly arise from the risks associated with a high market-cap cryptocurrency relying heavily on a limited number of centralized validators [79].

Interestingly, this centralization dilemma is not restricted to Ethereum alone. Bitcoin's network has faced similar challenges, making the centralization of Bitcoin a critical issue that resonates across the entire cryptocurrency market. This concern is particularly significant given that the vast majority of Bitcoin blocks are now produced by merely two mining pools [80]. For example, as shown in

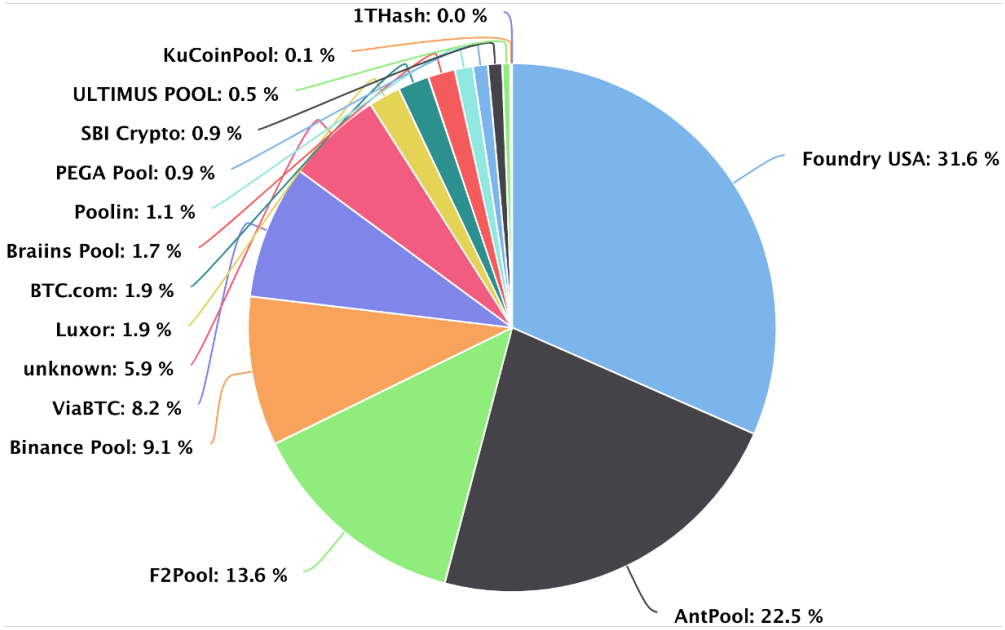

Figure 5.1: Ratio of Pool Distribution (calculated by blocks), 2023.04 2023.05 [1]

Figure 5.1, the global hash rate distribution of Bitcoin reveals that over half of the network's hash rate emanates from Foundry USA and Antpool. Specifically, Foundry's block production accounts for an estimated 31.6% of the entire network, and Antpool contributes about 22.5%. This concentration of block production within these two entities underscores an alarming level of centralization, with a consortium of interconnected companies effectively controlling half the network.

Moreover, the influence of these two entities extends beyond Bitcoin, contributing to a wider trend of centralization within the cryptocurrency industry. For example, Antpool operates mining pools for various other cryptocurrencies, including Litecoin (LTC), ZCash (ZEC), Bitcoin Cash (BCH), Ethereum Classic (ETC), and Dash (DASH). Similarly, Foundry provides enterprise staking

support for a range of cryptocurrencies, including Ethereum (ETH), Solana (SOL), Polkadot (DOT), Avalanche (AVAX), and Cosmos (ATOM).

In essence, these trends underscore the escalating concerns regarding centralization within the blockchain industry. The concentration of control within a select few entities not only compromises the security and trustless nature of these networks but also poses substantial risks to their stability and durability. Against this backdrop, it is imperative to investigate and develop mechanisms that can alleviate the centralization risks, bolster the decentralization of block production, and ensure the long-term viability of blockchain networks.

5.1.2 Verifiable Random Function

A Verifiable Random Function (VRF) [81] was first introduced by Micali, Rabin, and Vadhan in 1999, and since then, they've been extensively studied and implemented in various cryptographic and decentralized systems. The VRF is a cryptographic random function that maps inputs to verifiable pseudorandom outputs. It means that, unlike typical random functions, VRFs require a private key to compute the output, and anyone with the corresponding public key can verify the correctness of the output. This unique property makes VRFs particularly useful in decentralized systems and cryptographic protocols, where a way to prove the randomness and correctness of output can be critical. For example, they are particularly useful in proof-of-stake (PoS) blockchains for leader election and other randomized processes, because they provide a way to select nodes for participation in the consensus process in a way that is unpredictable, unbiased, and verifiable by all participants.

In a VRF, each input maps to a unique output and a unique proof. The owner of the private key can compute the output and the proof for any given input. Anyone else can use the public key, the input, the output, and the proof

to verify that the output and proof were correctly computed without needing the private key. Importantly, without proof, the output looks random to anyone who doesn't know the private key. VRFs consist of *keygen*, *prove*, and *verify* functions. First, as shown in equation 5.1, the ***keygen function*** is that generates private key SK and public key PK using an arbitrary input value k.

$$keygen(k) = (SK, PK) \tag{5.1}$$

Second, as shown in equation 5.2, the ***prove function*** uses the user's private key (sk) and input value (alpha) to calculate the VRF result value (beta) and the VRF result proof value (pi).

$$prove(sk, alpha) = (beta, pi) \tag{5.2}$$

Finally, as shown in equation 5.3, ***verify function*** verifies whether the generated value is the same as the VRF result value (beta) using the user's public key (pk) and the proof value (pi) of the VRF result. If the calculated value and the result value of VRF are the same, verification succeeds (True), otherwise verification fails (False).

$$verify(pk, alpha, beta, pi) = True\ or\ False \tag{5.3}$$

In summary, VRFs are a useful tool in cryptography protocols and decentralized systems, providing a source of randomness that is unpredictable, unbiased, verifiable, and resistant to manipulation. We use VRF to select validators (i.e., standing members) for consensus of block generation. By using the VRF, in selecting the validators, we can provide higher confidence and unbiased, verifiable, and resistant to manipulation.

5.2 Design and Implementation

To attain a higher degree of decentralization and performance, we introduce a novel consensus algorithm, the Proof of Double Committee (PoDC). The PoDC algorithm distinguishes validators into two categories: standing members and steering members. Standing members operate continuously and steering members are selected by a coordinator based on Verifiable Random Function (VRF) results. Both standing and selected steering members participate in the consensus process for block generation.

To do this, first, the proposed algorithm aims to decentralize the selection of a block proposer and block validators. For each round, a coordinator is selected among the standing members using a double hashing technique. Also, to prevent standing members collusion in block validation, we incorporate steering members, selected based on VRF, for changing the validator set every round. Therefore, with a double committee system, the proposed consensus algorithm ensures the random selection of validators in each round based on double hashing and VRF. This mechanism can offer superior randomness compared to the validator selection process in staking-based and Proof-of-Work (PoW) systems.

In terms of performance, the PoDC algorithm enables the participation of only 29 nodes (14 Standing Committee and 15 Steering Committee members) in the consensus process, even as the total number of nodes increases. There are several reasons for configuring the number of nodes. To thwart potential collusion within the standing committee, the proportion of steering members participating in the consensus is always maintained above 51%. It fulfills the existing Byzantine Fault Tolerance (BFT) consensus algorithm criteria of n = 3f + 1 (n: the number of total nodes / f: the number of faulty nodes), thus ensuring decentralization.

In addition, the PoDC is imperative that the number of steering committee candidates in the network exceeds 15. This requirement ensures that validators can be selected randomly for each round. Also, standing committee nodes, which are always operational, can be modified through governance votes, ensuring transparency in all nodes' operations.

5.2.1 Overview

The PoDC consists of the coordinator, standing members, steering members, and steering committee candidates:

- **Coordinator.** The coordinator is a block proposer who generates a block and proposes the block to each validator for block consensus. Coordinators are selected from standing members using double hashing.

- **Standing members** Standing members are steady nodes that consistently operate within the blockchain network and perpetually participate in block consensus. To serve as a standing member node, 44 million coins must be staked to ensure the stable operation of the consensus. If a standing member node operates abnormally, the staked amount is slashed. While rewards are provided to each standing member during block generation to ensure the stable operation of standing members.

- **Steering members** The steering members serve as a validator node to prevent collusion among the standing members. To participate as a steering member node, 110 thousand coins must be staked, and steering members for the next round are randomly selected based on VRF by the coordinator for each round. If a steering member node behaves abnormally, the staked amount is slashed. While, by participating in the consensus of block generation to prevent collusion of the standing members, they

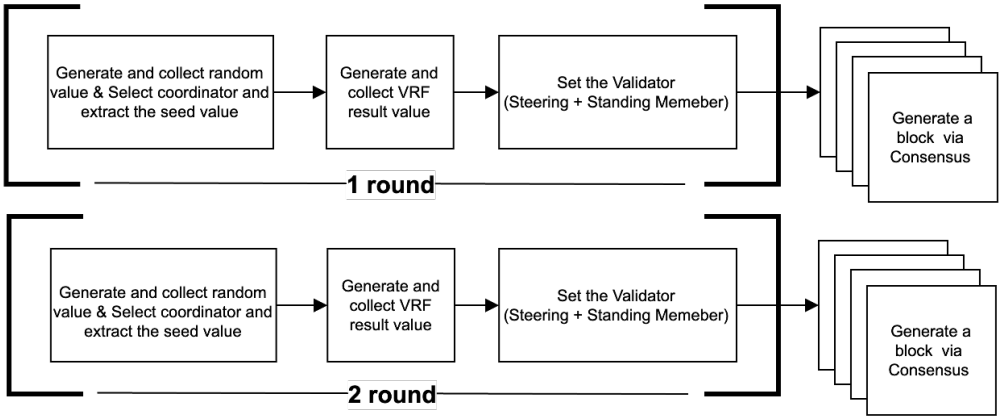

Figure 5.2: Overview of Proof of Double Committee

receive compensation upon block generation.

- **steering committee candidates** Steering committee candidates are nodes awaiting selection as steering members. 110,000 coins are staked, and the candidates should generate VRF results and deliver the VRF results to the coordinator for each round. Notice that a node serving as a steering member in the current round also is included as a steering committee candidate of the subsequent round.

Figure 5.2 shows the overall process of the proposed PoDC consensus algorithm which comprises a round. The round refers to the period for selecting the next validator. For example, during the first round, the coordinator and steering members for the second round are selected. The reason for selecting the validator group for the upcoming round during the current one is to accommodate network delays caused by communication across all nodes.

In each round, there are 3 steps. The first step is generating and collecting random values among standing members. And it selects the coordinator and extracts the seed value. The second step is generating VRF based on seed value

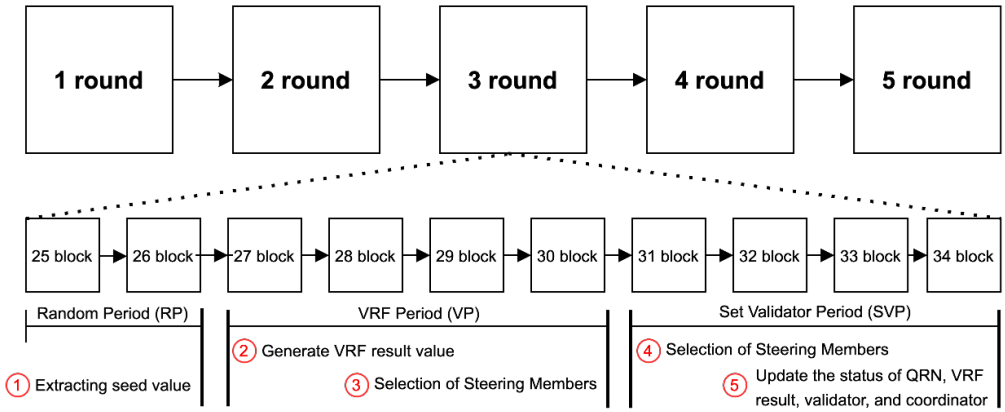

Figure 5.3: Detail of Proof of Double Committee

which is extracted in the first step and collecting VRF result values. The last step is setting the validators. The validators are 29 nodes and consist of standing members that are always in operation and randomly selected steering members.

Figure 5.3 shows the detailed process of a round in the proposed PoDC consensus algorithm. Each round is divided into three periods: the Random Period (RP), VRF Period (VP), and Setting Validators Period (SVP), each consisting of several blocks.

During the RP, standing members generate and propagate their own random values, which are used to determine the next coordinator and seed value. At the end of this period, all nodes derive a seed value based on the received random values and select the next coordinator (①). Following the RP, the VP commences. In this period, the steering committee candidates generate Verifiable Random Function (VRF) outcomes based on the seed derived from the RP (②). These VRF values are propagated to all nodes during the period. At the end of the VP, the coordinator selects 15 steering members based on the propagated VRF values (③). The final period, SVP, is dedicated to updating information about the standing members, selected steering members, random

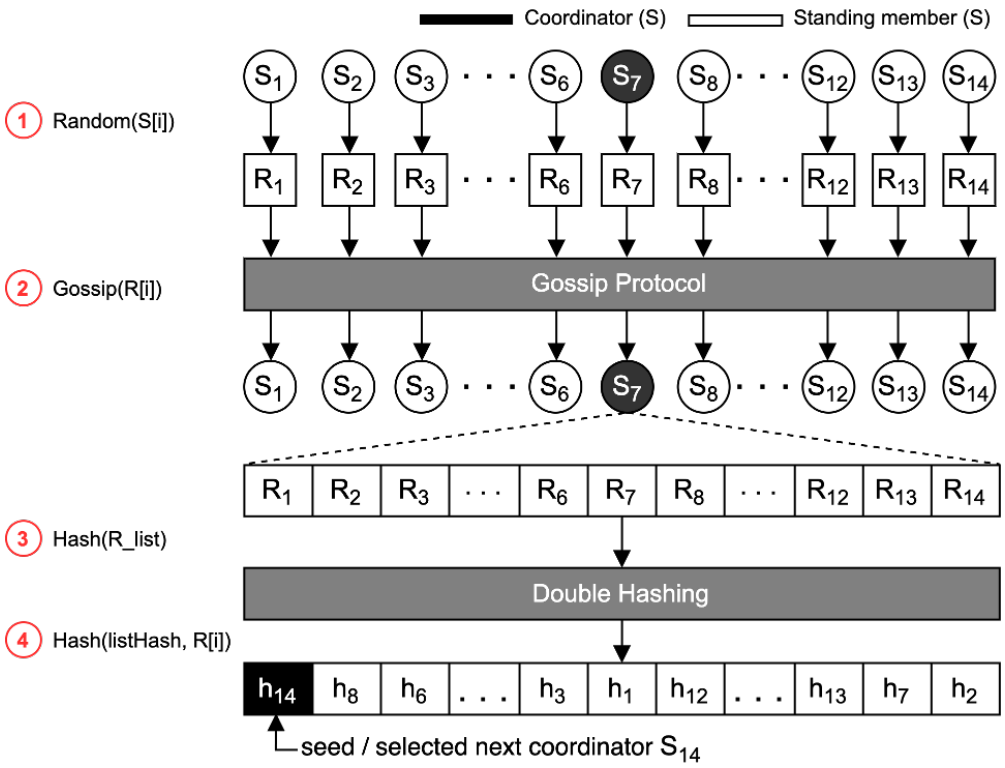

Figure 5.4: Process of the selecting next coordinator and current seed value in a round

values, and VRF results across all nodes. At the beginning of the SVP, the current round's coordinator disseminates information about the steering members selected during the VP to all nodes(④). Each node then verifies whether the data propagated by the current round's coordinator aligns with their data(⑤). Upon successful verification and at the end of the SVP, all nodes update the status of random values, VRF results, and the next round's validators and coordinator.

5.2.2 Selecting a coordinator and seed value

Figure 5.4 illustrates the process of selecting the next coordinator and seed value during a random period in the PoDC consensus algorithm. Initially, each standing member generates a random value, which is then disseminated to all other nodes in the network (①). This random value is signed by the standing member using their private key to ensure authenticity. The dissemination of the random values employs the gossip protocol, a peer-to-peer communication procedure modeled after the widespread propagation characteristic of epidemics (②). This protocol enables rapid and efficient distribution of the values across the network. Upon receiving the signed random value, all nodes verify its signature and confirm that the signer is indeed a registered standing member. If the verification is successful, the random value is added to a collective list of random values, referred to as the 'random list'.

Once all random values from the standing members are collected, a process of double hashing is performed based on the random list and each individual random value. The primary objective of this double-hashing process is to prevent any standing committee or coordinator from manipulating the random values. As shown in the equation 5.4, the initial step of the double hashing process concatenates all values in the random list, then subjects this concatenated string to a hashing operation (③). Subsequently, another hashing operation is performed on each individual random value in conjunction with the result of the initial hash (④). And, the result of this operation is the final hash value for each standing member. After then, all final hash values are sorted, and the highest value is designated as the seed value for the current round. And the standing member associated with this highest hash value is selected as the coordinator for the next round (⑤). This mechanism ensures a fair and un-

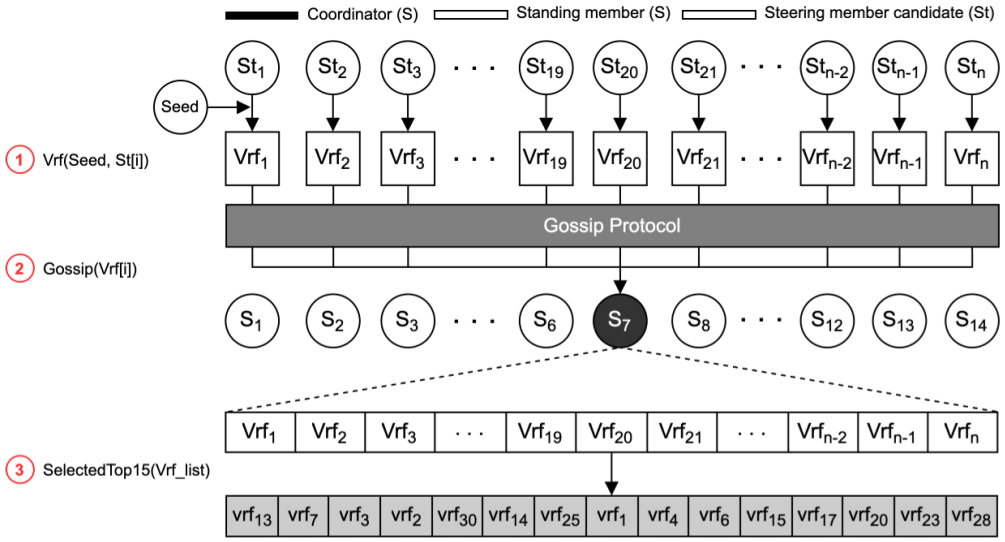

Figure 5.5: Process of selecting steering members

biased selection process, contributing to the decentralized nature of the PoDC consensus algorithm.

$$\begin{aligned} randomListHash &= Hash(R[1] + R[2] + R[3]... + R[14]) \\ h[i] &= Hash(randomListHash + R[i]) \quad \forall i \in R \\ seed &= NextMax(h\_list) \end{aligned} \tag{5.4}$$

5.2.3 Selecting steering members

Each steering committee candidate initiates the process by generating a VRF result. This result is generated using the seed value, which was selected during the preceding random period, and the candidate's private key. The VRF result consists of two key components: a random value and a proof value. The random value is a pseudo-random number that is generated based on the seed and the candidate's private key. The proof value, on the other hand, provides a method for verifying the legitimacy of the generated random value (①). Notably, due

to the deterministic nature of the VRF, the result is consistently the same for a given seed and private key pair. This deterministic attribute ensures the reliability of the generated values and underpins the fairness of the validator selection process.

Once the VRF result has been generated, it is then disseminated across the network. This is achieved by leveraging the efficient propagation capabilities of the gossip protocol. While the VRF result is not signed separately, the legitimacy of the steering committee candidate is verified using the proof. This proof value can be checked by each node using the public key of the candidate, affirming that the VRF result was indeed generated by a registered steering committee candidate. Upon successful verification, the random value is then added to the VRF list maintained by each node (②).

At the end time of the VRF period, the current coordinator performs a critical role. The coordinator assesses the VRF list and ranks the candidates based on the largest VRF random value. The top 15 ranked steering committee candidates, determined by this process, are then selected to participate as validators in the upcoming round (③). In summary, the VRF period represents a key phase of the PoDC consensus algorithm, facilitating a fair and verifiable process for the selection of steering committee candidates. This process ensures that the consensus mechanism remains robust and decentralized, reinforcing the integrity of the blockchain network.

5.2.4 Crash Fault Tolerant of the coordinator

The coordinator within the PoDC consensus algorithm plays a crucial role in both block generation and the selection of steering members for the subsequent round. However, due to factors such as network instability or node failure, there may be instances where the coordinator fails to fulfill its duties. This could

potentially disrupt the liveness of the blockchain and cause the blockchain to become inactive or unresponsive.

In PoDC, we incorporate a fail-safe algorithm to counter such situations. Each node in the network maintains a random list (i.e., *prev_random_list*) of the previous round's standing members which is used when the selection of the current round's coordinator. And, If the coordinator node fails to act, the timeout of the block proposer is triggered and all nodes then shift to a changing coordinator.

They update the status of the node with the second-largest hash value based on the *prev_random_list* to assume the role of the current round's coordinator. And, the standing committee member elevated to this role proceeds to create and propose a block as the current coordinator. The proposed block is then voted upon by all nodes in order to achieve Byzantine Fault Tolerance (BFT), satisfying the consensus criteria of 3f+1. Note that in cases where the former faulty coordinator re-establishes connection with the blockchain, it will receive the sink of the block. Upon receiving this, it will acknowledge the node with the second-highest hash value as the current coordinator. Subsequently, the former faculty coordinator will participate in the voting process for the block as a regular standing member, ensuring the continued smooth operation of the consensus process.

5.3 Evaluation

5.3.1 Experimental setup

We evaluate the distribution of block producers within a blockchain by contrasting it against Bitcoin, Cosmos (Tendermint), Algorand, and EOS. Considering that each blockchain functions under unique circumstances, our evaluation hinges on the block information from the main network of each respective

blockchain. We accumulated data from 10,000 blocks for each blockchain. The block information for each blockchain is as follows:

- **Bitcoin.** Information was collated from blocks 770,001 to 780,000.
- **Ethereum.** Information was collated from blocks 17,470,001 to 17,480,000.
- **Cosmos.** Information was collated from blocks 15,200,001 to 15,210,000.
- **Algorand.** Information was collated from blocks 28,940,001 to 28,950,000.
- **EOS.** Information was collated from 308,990,001 to 309,000,000.
- **PoDC.** Information was collated from 100,001 to 110,000.

In terms of validators that authenticate and vote for the validity of a block, Bitcoin is the absence of block validator information within the block. Therefore we do not evaluate validator variance.

In our evaluation, we use AWS's c2.xlarge environment running Ubuntu 20.04 LTS. Our proposed PoDC algorithm is based on the tendermint of the Cosmos, therefore, we compare with tendermint and PoDC while keeping the number of validators constant. We further scrutinized the performance by progressively increasing the number of nodes from 4 which is the minimum number of validators required for Byzantine Fault Tolerance (BFT) - up to 175, which corresponds to the current validator count in Cosmos.

5.3.2 Distribution of block proposers

Figure 5.6 displays the block generation count by individual miner addresses in the Bitcoin blockchain. During the generation of 10,000 blocks, we observed that 129 miner addresses were involved. Of these, the address *bc1qxhmduf...* contributed to the generation of 3,296 blocks, and *38XnPvu9Pm...* produced

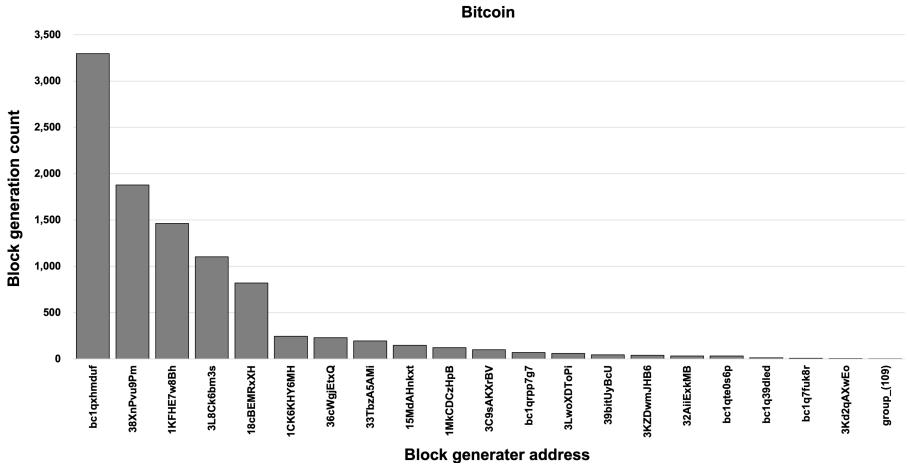

Figure 5.6: The block generation count of a proposer (miner) in Bitcoin blockchain

1,878 blocks. Hence, these two wallet addresses were responsible for 5,174 blocks (51.74%) of all blocks generated, surpassing 51% of the total block generation rate. Moreover, four wallet addresses, including *1KFHE7w8Bh...* (1,462 blocks) and *3L8Ck6bm3s...* (1,102 blocks), both of which exceeded 1,000 blocks in a generation, collectively produced 7,738 (77.38%) blocks. This statistic represents only 3.1% of the total 129 wallet addresses, underscoring that a majority of blocks were generated by a small subset of wallets. Because Bitcoin employs the Proof of Work (PoW) consensus algorithm for block generation, mining pools are predominantly responsible for block production. Consequently, Bitcoin's blockchain faces significant challenges in achieving true decentralization.

Figure 5.7 displays the block generation count by individual miner addresses in the Ethereum blockchain. During the generation of 10,000 blocks, we observed that 665 miner addresses were involved. Of these, the address *0x1f9090a...* contributed to the generation of 1,810 blocks, *38XnPvu9Pm...* produced 1,252 blocks, *0xdafea49...* produced 1,189 blocks, and *0x388c818...* produced 1,095 blocks. Hence, these four wallet addresses were responsible for

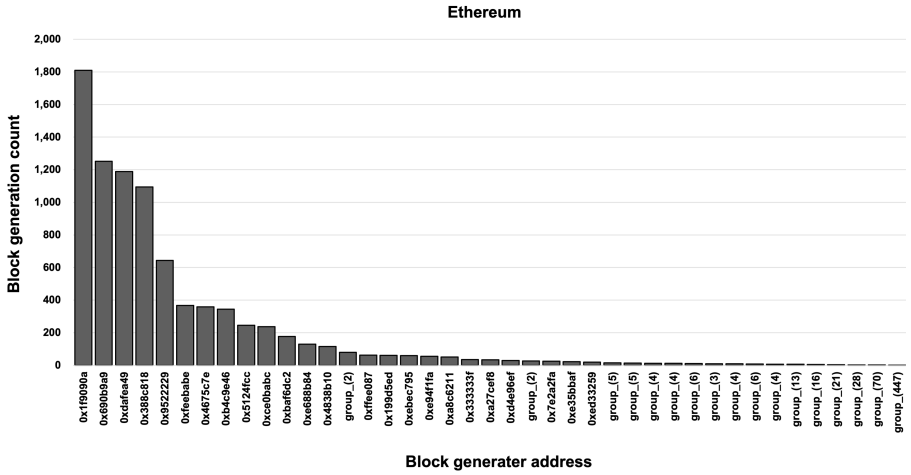

Figure 5.7: The block generation count of a proposer (miner) in Ethereum blockchain

5,346 blocks (53.46%) of all blocks generated, surpassing 51% of the total block generation rate. This statistic represents only 0.6% of the total 665 wallet addresses, underscoring that a majority of blocks were generated by a small subset of wallets. Because Ethereum employs the Proof of Work (PoW) consensus algorithm for block generation, mining pools are predominantly responsible for block production. Also, Ethereum employs a Proof of Stake (PoS) consensus algorithm for block finalization. It means that finality in Ethereum is provided by utilizing PoS, but it ultimately only proceeds with voting for blocks created by miners. Consequently, Ethereum's blockchain also faces significant challenges in achieving true decentralization.

Figure 5.8 illustrates the distribution of block generation by addresses in the Algorand blockchain. From a total of 10,000 blocks, 66 unique addresses contributed as block proposers. The address identified as *OFB2SM...* was the most productive, generating 710 blocks, accounting for 7.1% of the total blocks. Additionally, the top 12 addresses collectively produced 5,302 blocks or 53.02%

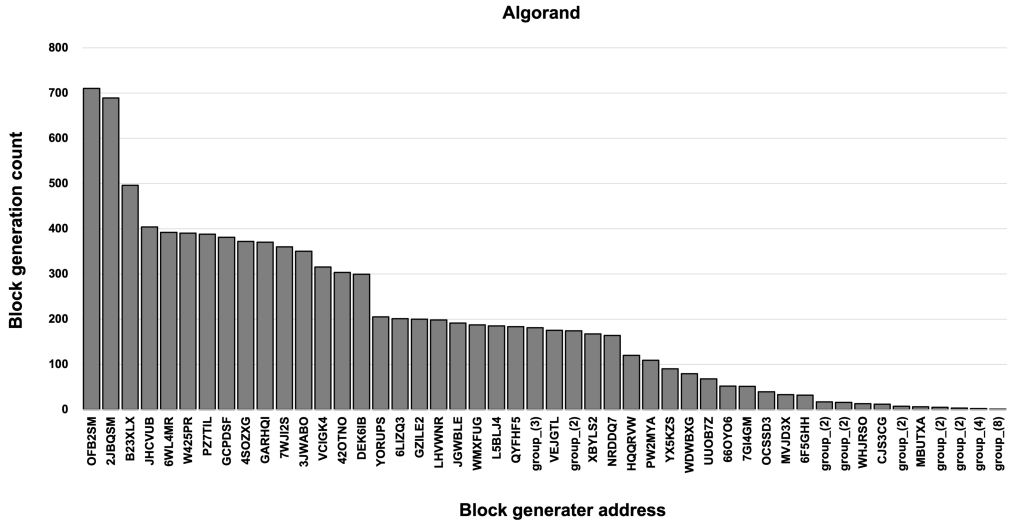

Figure 5.8: The block generation count of a proposer in Algorand blockchain

of the total. These addresses represent 18.18% of the 66 unique addresses contributing to block generation. Furthermore, the top 23 addresses, making up 34.84% of all unique addresses, generate 7,769 (77.69% of the total)blocks. This indicates that a significant portion of block generation is concentrated among a minority of addresses.

Algorand exploits a Verifiable Random Function (VRF) to select block proposers, designed to give every participant in the network an equal opportunity to generate blocks, reinforcing blockchain decentralization. However, it also adopts proof of stake, so it does not provide a fair opportunity for block creators. Similar to Bitcoin, it appears that the decentralization of the Algorand blockchain is threatened due to the dominance of block generation by addresses holding a larger share.

Figure 5.9 illustrates the count of block generations per validator address on the Cosmos blockchain. Out of the total 10,000 blocks generated, 174 distinct addresses participated as block proposers. The address *D68EEC...* was most

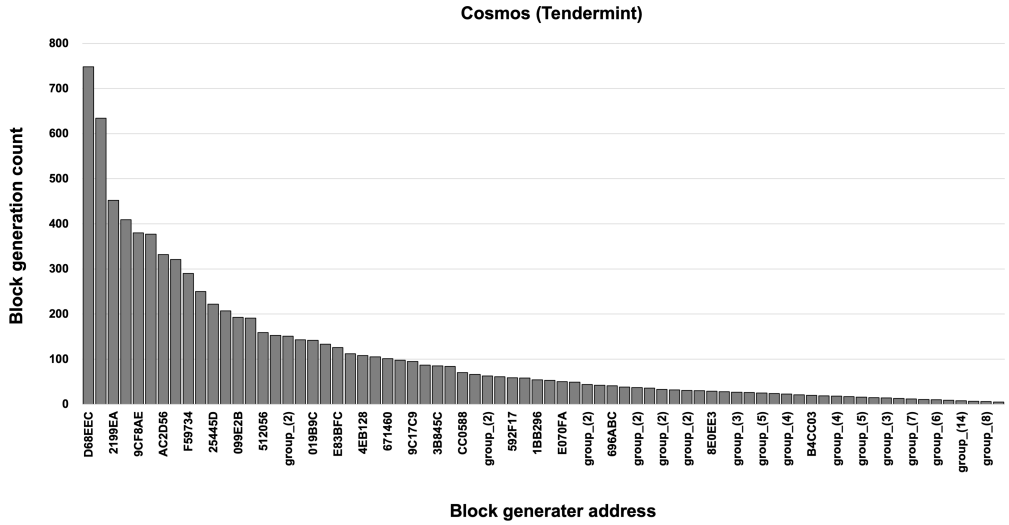

Figure 5.9: The block generation count of a proposer in Cosmos blockchain

active, creating 748 blocks, accounting for 7.48% of the total blocks generated. Furthermore, the top 15 wallet addresses together produced 5,165 blocks, which equates to 51.65% of all the blocks, indicating a significant majority of block generation by these top wallets. This group of 15 represents merely 8.62% of the total 174 wallet addresses, further emphasizing that a small fraction of wallets was responsible for the majority of block production. A more stark observation is that the top 41 wallet addresses (representing 23.56% of the total addresses) were responsible for 7,701 blocks, or 77.01% of the total blocks.

Cosmos uses a round-robin mechanism for selecting block proposers, where the privilege of block production is systematically passed on to each validator following a defined sequence. However, this system has limitations in protecting the network from Byzantine attacks. To solve this issue, Cosmos implements vote weights, determined by the number of Cosmos tokens a validator holds, leading validators with more tokens to exert more influence. In such a setting, the decentralization of the blockchain can be threatened if wallets with higher

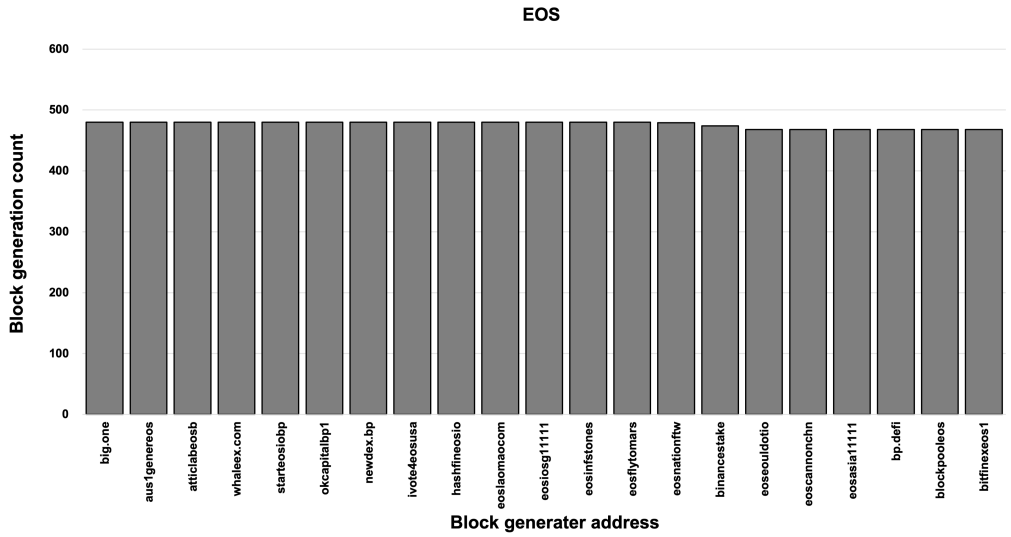

Figure 5.10: The block generation count of a proposer in EOS blockchain

stakes dominate block production, as observed with other blockchains.

Figure 5.10 displays the block generation count per miner address on the EOS blockchain. A total of 10,000 blocks were generated, with 21 addresses participating as block proposers. Out of these, 13 addresses each produced 480 blocks, which each account for 4.8% of the total block generation. Moreover, the top 11 wallet addresses collectively generated 5,280 blocks (52.8%), surpassing 51% of the overall block generation rate. These 11 wallets make up 52.38% of all 21 wallet addresses, indicating a relatively even distribution of block production.

The reason behind this trend is that EOS selects 21 fixed validators based on stake, and employs a round-robin mechanism to determine block generators. The order of block generation is transferred to each validator in turn according to specific rules. Although this approach allows for a more equal distribution of block creators compared to other blockchains, it has limitations in protecting the network from Byzantine attacks due to its reliance on fixed validators.

Figure 5.11 illustrates the block generation count per block proposer address

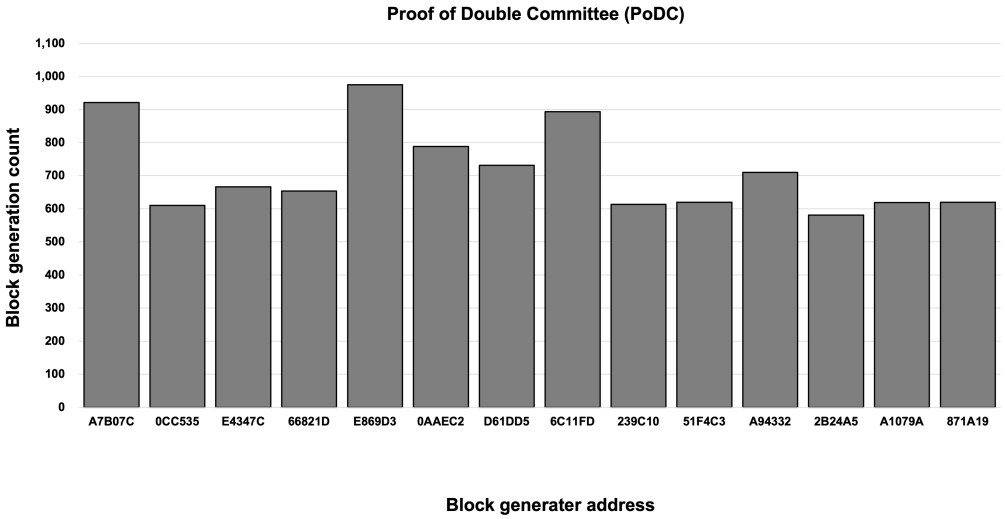

Figure 5.11: The block generation count of a proposer (coordinator) in PoDC blockchain

in the PoDC system. In generating a total of 10,000 blocks, 14 addresses participated as block proposers. Of these, the *A7B07C...* address created the most blocks, with a count of 921 out of 10,000 blocks. This corresponds to 9.21% of the total block generation. Furthermore, the top 7 wallet addresses collectively produced 5,280 blocks (53.45%), surpassing the 51% threshold of the overall block generation rate. These top 7 wallet addresses constitute 50% of all 14 wallet addresses, indicating a relatively even distribution of block creation.

This pattern can be attributed to the PoDC system's approach. 14 fixed validators (i.e., standing members) are selected based on a fixed stake(44 million), and block generation order is determined by a double-hashing rule. Unlike other blockchains like Bitcoin, Cosmos, and Algorand, this allows for an equitable distribution of block generators. Furthermore, block generators are randomly chosen, enhancing network protection against Byzantine attacks.

Figure 5.12 and Figure 5.13 indicate the sequence of block generators. In the case of EOS, as shown in Figure 5.12, the order of the block proposer is

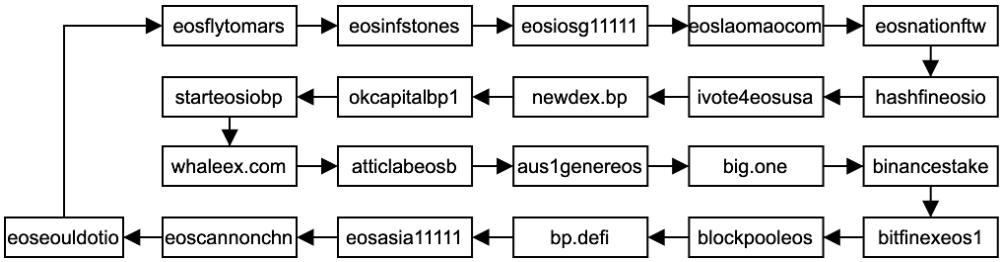

Figure 5.12: The block proposer order in EOS blockchain

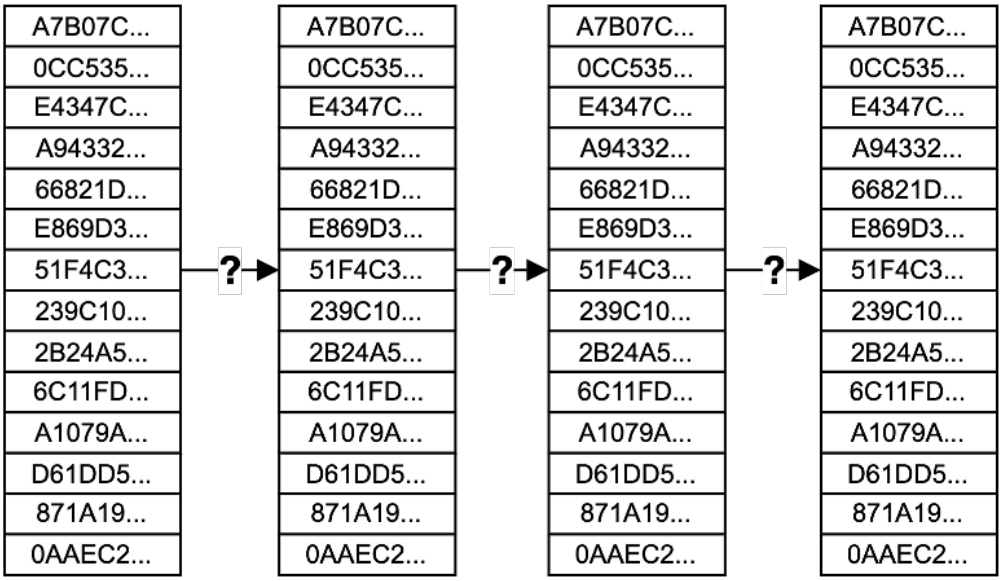

Figure 5.13: The block proposer order in PoDC

changed by 12 blocks, and it can be seen that it is changed in a certain order. In other words, it can be seen that EOS creates blocks in the form of round robin.

The round-robin method is a method in which the block generation order is determined sequentially for each node. It may seem that the rules are straightforward and that all players in the system are given an equal chance. However, this arrangement is vulnerable to Byzantine attacks. The reason is that byzan-

tine attacks can occur when malicious nodes gain a majority of the system. These malicious nodes can work together to harm or manipulate the network. For example, a malicious node selected as a block producer may include false information in a block or not create a block. Since the order of block generation in the round-robin method is pre-determined, malicious nodes can know the order of block generation. Therefore, malicious nodes can use this information to perform Byzantine attacks. For example, since malicious nodes know the order in which certain nodes generate blocks, they can attack the network in that order or attack nodes to disrupt block creation.

Meanwhile, in the case of PoDC as shown in Figure 5.13, the order of the block proposer (coordinator) is changed by 12 blocks, but it is not a deterministic block proposer. Therefore, it can provide more security than EOS while providing fair block producers. However, PoDC can also identify the next block producer up to 12 blocks in advance. Based on the average block generation time of 5 seconds, this can take about 1 minute. So, to solve this issue, the number of blocks in each period can be adjusted, and the identification of the following block creator can be reduced to at least 3 blocks (15 seconds) depending on the situation.

Figure 5.14 depicts the count of block generation by a proposer (coordinator), varying with different standing members. The top figure shows that with 14 standing members, the average number of blocks generated within 10,000 blocks is 714, translating to an average block generation probability of 7.14%. As shown in the figure, 19A481 generates 803 blocks, accounting for 8.03%, and 499A4B produces 485 blocks, making up 4.85%. This variance is attributed to the random selection of proposers via double hashing. Moreover, as exhibited in the middle and bottom diagrams, when the standing members' count escalates to 21 and 30 respectively, the average block generation drops to 476 (4.76%)

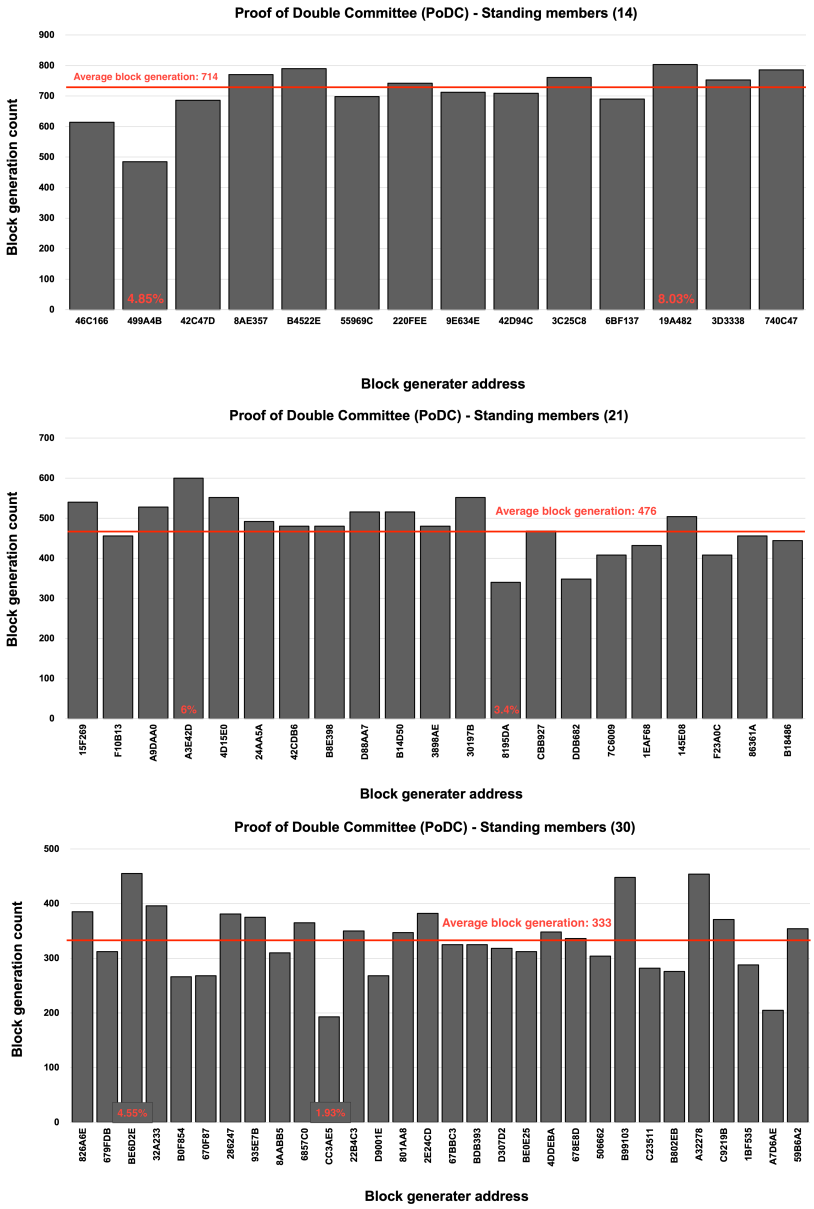

Figure 5.14: The block generation count of a proposer (coordinator) with different standing members in PoDC

and 333 (3.33%). This implies that an increase in the number of standing members can decrease the weight of block generation. If the number of standing

members is set with 21 nodes, as displayed in Figure 5.10, the percentage of blocks a single node can generate spans from a minimum of 3.4% to a maximum of 6%, indicating a similar level of fairness. Furthermore, increasing the standing members to 30 leads to each node producing a minimum of 1.93% and a maximum of 4.55% of blocks. This trend suggests a more stable block generation ratio with fewer nodes than Cosmos, as depicted in Figure 5.9.

Distribution of validators.

Figure 5.15 shows the probability of being selected as a validator in 10,000 blocks, contingent on the number of steering committee candidates. We maintain the number of standing members at 14 and select 15 steering members for the steering member to account for more than 51% of the total number of validators. As shown in the top figure of Figure 5.15, if the number of steering committee candidates is set to 15, all candidates are selected as steering members, participating in all blocks as validators. However, due to the high potential for collusion in such a scenario, it is advisable to expand the number of participating steering members.

Therefore, as shown in the middle and bottom figures of Figure 5.15, when the number of steering committee candidates is increased to 46 and the 16,1 participation rate of each node as a validating member would significantly decrease, and then the random selection process could mitigate collusion. The number of steering committee candidates is augmented to 46, approximately three times the number of selected steering members, and the probability of a single steering committee candidate participating as a validator is on average 32.6%. The range of probabilities shows from a minimum of 29.33% to a maximum of 38.87%.

Furthermore, selecting 161 steering committee candidates is aimed at matching the current composition of validators in Cosmos, which consists of 175

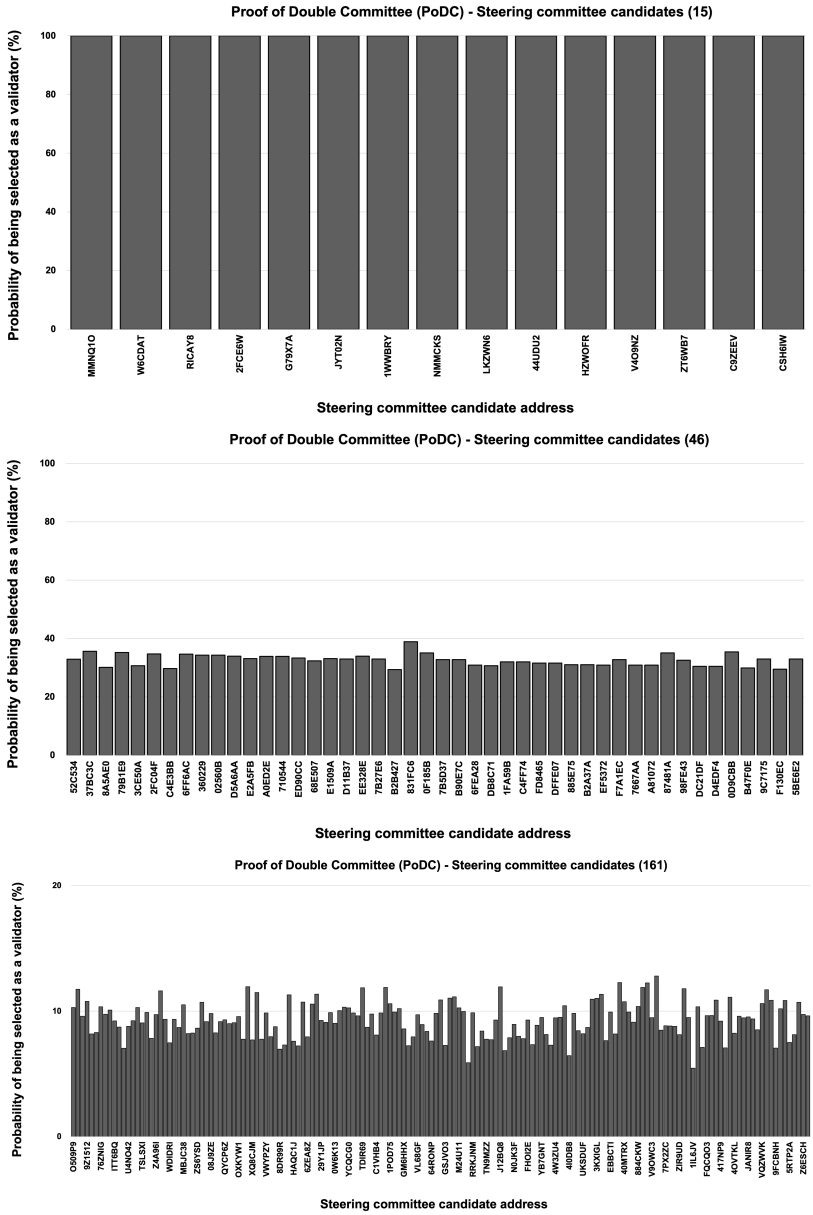

Figure 5.15: Probability of validator selection of steering committee candidates in PoDC

members. In this case, the average probability for a single steering committee candidate to participate as a validator is 9.31%. The range of probabilities

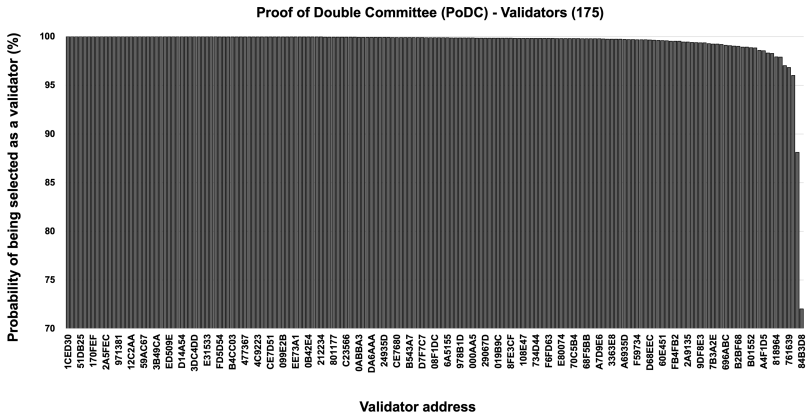

Figure 5.16: Probability of validator selection in cosmos(tendermint)

shows from a minimum of 5.46% to a maximum of 12.8%. This can be seen as having the same effect as all validators participating and reaching a consensus, as shown in Figure 5.16. Moreover, since only 29 nodes are required for consensus, this PoDC algorithm is faster in achieving consensus compared to the 175 nodes in Cosmos. Lastly, since not all validators are fixed and can be randomly selected, stability is ensured. It means that, in the case of Cosmos, only the top 175 validators are eligible to participate and it almost does not be changed. Therefore, increasing the likelihood of collusion. However, our proposed Proof of Decentralized Consensus (PoDC) allows participation as steering member candidates through minimal staking and random selection of steering members. Through this approach, we can form a more diverse set of validators.

5.4 Summary

We investigate blockchain consensus algorithms, which serve as the mechanism enabling all network participants to reach an agreement on the validity of transactions and the blocks to be added to the chain. And we found that most consensus algorithms suffer from a centralization problem and performance issue. To

overcome these issues, we propose a new consensus algorithm which is proof of double committee(PoDC). We implement PoDC using the random selection of a fixed number of validators via double hashing and verifiable random function (VRF) and evaluate the algorithm's degree of decentralization and performance for each blockchain. Our experimental results show that PoDC provides high decentralization rate and high performance than other existing blockchains.

Chapter 6

Conclusion

6.1 Discussion

This article provides an extensive exploration into several core elements of blockchain technology, specifically focusing on enhancements in wallet security, enabling SQL query, and block generation consensus algorithm. Each of these facets contributes significantly to the blockchain ecosystem, and the advancements proposed in this research have the potential to greatly influence the future development of this technology.

The article introduces a novel wallet model that incorporates the Threshold Elliptic Curve Digital Signature Algorithm (T-ECDSA) and Bloom filters to create a multi-signature wallet. This is an impressive leap in addressing existing security vulnerabilities that have been plaguing blockchain wallets. However, while the proposed wallet model shows promise in lab tests, real-world applications might present unforeseen challenges. Further research and testing in diverse, real-world environments are necessary to ascertain its performance

under various conditions and to identify potential limitations. A comprehensive analysis of its resilience to different types of security threats should also be conducted to ensure robustness.

The introduction of a mechanism to incorporate SQL query operations into blockchain systems is another significant development. By embedding a relational database within an Ethereum-based blockchain system, it becomes possible to execute SQL queries directly. However, while this enhancement promises to streamline the operation of the blockchain system and improve efficiency, potential challenges need to be addressed. The integration process must ensure data consistency and security. Further studies could explore how to effectively manage potential inconsistencies or discrepancies in data between the blockchain and the embedded database.

Lastly, the dual committee proof (PoDC) consensus mechanism introduced in this article offers a promising solution to centralization issues related to traditional consensus algorithms like Proof of Work (PoW) and Tendermint. However, like any new consensus mechanism, the long-term stability of this method will require rigorous scrutiny. Extensive testing and performance analyses under various scenarios would provide a comprehensive understanding of the overall benefits and potential pitfalls of this approach.

6.2 Summary

Blockchain facilitates dependable transactions within a decentralized environment, making various user contracts possible without the need for an intermediary authority. However, with the increasing interest in and usage of blockchain technology, several problems have come to light. These include wallet security vulnerabilities, user manipulation of transaction execution orders, and issues regarding blockchain centralization and scalability linked to consensus algorithms.

Therefore, enhancing blockchain security and performance has become a priority. In this article, we concentrate on three aspects to bolster the security of the blockchain system and enhance its performance: wallet security, transaction execution order determination, and block generation consensus algorithm.

Firstly, we propose an efficient multi-signature wallet, utilizing Threshold Elliptic Curve Digital Signature Algorithm (T-ECDSA) and Bloom filters. This wallet offers improved performance, storage efficiency, and privacy, addressing the security vulnerabilities of blockchain wallets. It also enhances verification performance and reduces transaction size without altering the blockchain protocol.

Secondly, we introduce a mechanism to incorporate SQL query operations within a blockchain system. It allows us to embed a relational database within an Ethereum-based blockchain system, facilitating SQL queries. Consequently, we can perform range queries for smart contracts without the need for user-defined data structures, and we reduce management costs for regular transactions by eliminating the necessity for an external database. Our experimental results show that our proposed system can improve the search performance of smart contracts up to approximately 22 times compared to existing systems. Moreover, our system exhibits comparable search performance to existing systems that incorporate an external database in the case of regular transactions.

Finally, we introduce a dual committee proof (PoDC) consensus mechanism within the blockchain consensus. This mechanism divides validators into standing members and steering members, randomly selecting validators through a double-hashing process and a Verifiable Random Function (VRF). Our method alleviates centralization issues associated with existing consensus algorithms like Proof of Work (PoW) and Tendermint, and it also improves performance in blockchains with a fixed number of validators. In conclusion, this article

breaks significant ground in the field of blockchain technology, tackling several pressing challenges with innovative solutions. The potential enhancements it suggests in terms of security, efficiency, and decentralization not only hold promise for immediate implementation but also open the door for further research. By continuing to explore and develop these concepts, we can build a more secure, efficient, and fair blockchain ecosystem, capable of meeting the demands of an increasingly digital future.

Bibliography

[1] "Bitcoin pool distribution." `https://btc.com/stats/pool?pool_mode=month`, 2023.

[2] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," *International journal of web and grid services*, vol. 14, no. 4, pp. 352–375, 2018.

[3] F. Tschorsch and B. Scheuermann, "Bitcoin and beyond: A technical survey on decentralized digital currencies," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2084–2123, 2016.

[4] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[5] S. S. Sarmah, "Understanding blockchain technology," *Computer Science and Engineering*, vol. 8, no. 2, pp. 23–29, 2018.

[6] D. Perez and B. Livshits, "Smart contract vulnerabilities: Vulnerable does not imply exploited.," in *USENIX Security Symposium*, pp. 1325–1341, 2021.

[7] J. Herrera-Joancomartí and C. Pérez-Solà, "Privacy in bitcoin transactions: new challenges from blockchain scalability solutions," in *Modeling Decisions for Artificial Intelligence: 13th International Conference, MDAI 2016, Sant Julià de Lòria, Andorra, September 19-21, 2016. Proceedings 13*, pp. 26–44, Springer, 2016.

[8] I. Abraham, D. Malkhi, *et al.*, "The blockchain consensus layer and bft," *Bulletin of EATCS*, vol. 3, no. 123, 2017.

[9] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi, "Towards scaling blockchain systems via sharding," in *Proceedings of the 2019 international conference on management of data*, pp. 123–140, 2019.

[10] A. E. Gencer, S. Basu, I. Eyal, R. Van Renesse, and E. G. Sirer, "Decentralization in bitcoin and ethereum networks," in *Financial Cryptography and Data Security: 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26–March 2, 2018, Revised Selected Papers 22*, pp. 439–457, Springer, 2018.

[11] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016.

[12] "Wikipedia: Multisignature." `https://en.wikipedia.org/wiki/Multisignature`, 2020.

[13] "Multisigwallet." `https://github.com/Gnosis/MultiSigWallet`, 2019.

[14] "Bitgo." `https://www.bitgo.com/`, 2014.

[15] "Etherscan.io," 2018. `https://etherscan.io`.

[16] "Is it possible to access storage history from a contract in solidity?." `https://ethereum.stackexchange.com/questions/11545/is-it-possible-to-access-storage-history-from-a-contract-in-solidity`, 2016.

[17] F. A. Pratama and K. Mutijarsa, "Query support for data processing and analysis on ethereum blockchain," in *2018 International Symposium on Electronics and Smart Devices (ISESD)*, pp. 1–5, IEEE, 2018.

[18] "The graph," 2020. `https://thegraph.com`.

[19] M. Du, Q. Chen, and X. Ma, "Mbft: A new consensus algorithm for consortium blockchain," *IEEE Access*, vol. 8, pp. 87665–87675, 2020.

[20] E. Buchman, *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, 2016.

[21] N. Alzahrani and N. Bulusu, "Towards true decentralization: A blockchain consensus protocol based on game theory and randomness," in *International conference on decision and game theory for security*, pp. 465–485, Springer, 2018.

[22] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th symposium on operating systems principles*, pp. 51–68, 2017.

[23] A. Baliga, "Understanding blockchain consensus models," *Persistent*, vol. 4, no. 1, p. 14, 2017.

[24] S. Gupta and M. Sadoghi, "Blockchain transaction processing," *arXiv preprint arXiv:2107.11592*, 2021.

[25] U. Rahardja, A. N. Hidayanto, N. Lutfiani, D. A. Febiani, and Q. Aini, "Immutability of distributed hash model on blockchain node storage," *Sci. J. Informatics*, vol. 8, no. 1, pp. 137–143, 2021.

[26] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International journal of information security*, vol. 1, pp. 36–63, 2001.

[27] "Elliptic-curve cryptography." `https://en.wikipedia.org/wiki/Elliptic-curve_cryptography`, 2020.

[28] A. A. Imem, "Comparison and evaluation of digital signature schemes employed in ndn network," *arXiv preprint arXiv:1508.00184*, 2015.

[29] "Bip: 16." `https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki`, 2012.

[30] "Transactions," 2022. `https://ethereum.org/ko/developers/docs/transactions/`.

[31] V. Buterin, "A next-generation smart contract and decentralized application platform," 2014.

[32] N. Szabo, "The idea of smart contracts," 1997.

[33] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," 2014.

[34] M. S. Chishti, F. Sufyan, and A. Banerjee, "Decentralized on-chain data access via smart contracts in ethereum blockchain," *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 174–187, 2021.

[35] "Leveldb," 2019. `https://github.com/google/leveldb`.

[36] D. Mingxiao, M. Xiaofeng, Z. Zhe, W. Xiangwei, and C. Qijun, "A review on consensus algorithm of blockchain," in *2017 IEEE international conference on systems, man, and cybernetics (SMC)*, pp. 2567–2572, IEEE, 2017.

[37] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 3–16, 2016.

[38] C. T. Nguyen, D. T. Hoang, D. N. Nguyen, D. Niyato, H. T. Nguyen, and E. Dutkiewicz, "Proof-of-stake consensus mechanisms for future blockchain networks: fundamentals, applications and opportunities," *IEEE Access*, vol. 7, pp. 85727–85745, 2019.

[39] X. Hao, L. Yu, L. Zhiqiang, L. Zhen, and G. Dawu, "Dynamic practical byzantine fault tolerance," in *2018 IEEE conference on communications and network security (CNS)*, pp. 1–8, IEEE, 2018.

[40] D. Larimer, "Dpos consensus algorithm-the missing white paper," *Bitshare whitepaper*, 2017.

[41] S. Goldfeder, R. Gennaro, H. Kalodner, J. Bonneau, J. A. Kroll, E. W. Felten, and A. Narayanan, "Securing bitcoin wallets via a new dsa/ecdsa threshold signature scheme," in *et al.*, 2015.

[42] R. Gennaro, S. Goldfeder, and A. Narayanan, "Threshold-optimal dsa/ecdsa signatures and an application to bitcoin wallet security," in *International Conference on Applied Cryptography and Network Security*, pp. 156–174, Springer, 2016.

[43] R. Gennaro and S. Goldfeder, "Fast multiparty threshold ecdsa with fast trustless setup," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1179–1194, 2018.

[44] "Ledger." `https://www.ledger.com/`, 2020.

[45] "Trezor." `https://trezor.io/`, 2020.

[46] "Keepkey." `https://shapeshift.com/keepkey`, 2020.

[47] "Etherchain." `https://www.etherchain.org`, 2018.

[48] "Ethstats," 2018. `https://ethstats.net`.

[49] B. Platz, A. Filipowski, and K. Doubleday, "Flureedb: a practical decentralized database," 2017.

[50] T. McConaghy, R. Marques, A. Müller, D. De Jonghe, T. McConaghy, G. McMullen, R. Henderson, S. Bellemare, and A. Granzotto, "Bigchaindb: a scalable blockchain database," *white paper, BigChainDB*, 2016.

[51] Y. Li, K. Zheng, Y. Yan, Q. Liu, and X. Zhou, "Etherql: a query layer for blockchain system," in *International Conference on Database Systems for Advanced Applications*, pp. 556–567, Springer, 2017.

[52] "Ethereumj." `https://github.com/ethereum/ethereumj`, 2018.

[53] Z. Peng, H. Wu, B. Xiao, and S. Guo, "Vql: Providing query efficiency and data authenticity in blockchain systems," in *2019 IEEE 35th International Conference on Data Engineering Workshops (ICDEW)*, no. 7, pp. 1–6, IEEE, 2019.

[54] S. Bragagnolo, H. Rocha, M. Denker, and S. Ducasse, "Ethereum query language," in *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*, pp. 1–8, 2018.

[55] A. Goldberg and D. Robson, *Smalltalk-80: the language and its implementation*. Addison-Wesley Longman Publishing Co., Inc., 1983.

[56] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," tech. rep., Manubot, 2019.

[57] M. Conti, E. S. Kumar, C. Lal, and S. Ruj, "A survey on security and privacy issues of bitcoin," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3416–3452, 2018.

[58] J. Kwon and E. Buchman, "Cosmos whitepaper," *A Netw. Distrib. Ledgers*, p. 27, 2019.

[59] B. Xu, D. Luthra, Z. Cole, and N. Blakely, "Eos: An architectural, performance, and economic analysis," *Retrieved June*, vol. 11, p. 2019, 2018.

[60] A. Ngunyi, S. Mundia, and C. Omari, "Modelling volatility dynamics of cryptocurrencies using garch models," 2019.

[61] "Kucoin hack: 280m dollars stolen not 150m dollars, projects take actions to save funds- community reacts. will kucoin recover all?." `https://thecurrencyanalytics.com/crypto-exchanges/kucoin-hack-280m-stolen-not-150m-projects-take-actions-to-save-funds-community-reacts-will-kucoin-recover-all-19577.php`, 2020.

[62] "Upbit is the seventh major crypto exchange hack of 2019." `https://www.coindesk.com/markets/2019/11/27/upbit-is-the-seventh-major-crypto-exchange-hack-of-2019/`.

[63] "Hackers steal over 40 million dollars worth of bitcoin from one of the world's largest cryptocurrency exchanges." https://www.cnbc.com/2019/05/08/binance-bitcoin-hack-over-40-million-of-cryptocurrency-stolen.html, 2019.

[64] V. Shoup, "Practical threshold signatures," in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 207–220, Springer, 2000.

[65] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[66] "What is a blockchain address?." https://wirexapp.com/help/article/what-is-a-blockchain-address-0068, 2020.

[67] C. Rackoff and D. R. Simon, "Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack," in *Annual International Cryptology Conference*, pp. 433–444, Springer, 1991.

[68] "Commitment scheme." https://en.wikipedia.org/wiki/Commitment_scheme, 2020.

[69] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[70] "bitcoin-multisig." https://github.com/soroushjp/go-bitcoin-multisig, 2014.

[71] "Bitcoin fee calculator estimator." https://www.buybitcoinworldwide.com/fee-calculator, 2020.

[72] "Eth gas station." https://ethgasstation.info/calculatorTxV.php, 2020.

[73] "Opensea statistics 2023: How many users does opensea have?." https://thesmallbusinessblog.net/opensea-statistics/, 2023.

[74] G. Caldarelli, "Understanding the blockchain oracle problem: A call for action," *Information*, vol. 11, no. 11, p. 509, 2020.

[75] "what is the difference between embedded database and ordinary database like mysql or oracle." https://goo.gl/oV9x7b, 2018.

[76] "Wikipedia: Embedded database." https://en.wikipedia.org/wiki/Embedded_database, 2018.

[77] "Sqlite." https://www.sqlite.org/index.html, 2018.

[78] "Apache jmeter," 2022. https://jmeter.apache.org/.

[79] X. Xu, I. Weber, M. Staples, L. Zhu, J. Bosch, L. Bass, C. Pautasso, and P. Rimba, "A taxonomy of blockchain-based systems for architecture design," in *2017 IEEE international conference on software architecture (ICSA)*, pp. 243–252, IEEE, 2017.

[80] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "Sok: Research perspectives and challenges for bitcoin and cryptocurrencies," in *2015 IEEE symposium on security and privacy*, pp. 104–121, IEEE, 2015.

[81] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, pp. 120–130, IEEE, 1999.

Abstract

블록체인 시스템은 신뢰할 수 없는 탈중앙화된 환경에서 안정적인 사용자 간의 자금 전송을 실현하며, 스마트 컨트랙트의 활용을 통해 중간 신뢰 기관 없이 다양한 계약을 체결할 수 있도록 하는 구조를 가지고 있다. 그러나, 블록체인의 확장성과 사용 증가에 따른 다양한 중앙 집중화 문제가 도출되고 있다. 특히, 단일 서명 지갑의 보안 취약성, 블록체인 거래 내역 조회 성능의 부재 및 외부 데이터베이스의 단일 지점 실패, 그리고 합의 알고리즘에서의 검증자 중앙화 경향 등이 주요 문제로 지적되고 있다. 이러한 이유로, 블록체인에서의 중앙화 문제를 극복하고 보안 강화와 성능을 향상시키는 것이 블록체인 시스템에서의 중요한 이슈로 부상하고 있다. 본 연구에서는 이 문제들을 극복하기 위해, 단일 서명 지갑의 보안 강화, 외부 데이터베이스 의존성 해결을 통한 거래 검색 성능 향상, 그리고 합의 알고리즘의 중앙화 이슈를 해결하는 측면에 초점을 맞춘다.

첫 번째로, 블록체인 단일 서명 지갑의 보안 취약성을 해결하기 위해, 임계 타원곡선 전자 서명 알고리즘(T-ECDSA)과 블룸 필터를 결합한 효율적인 다중 서명 지갑을 제안한다. 이 지갑은 블록체인 프로토콜을 변경하지 않고도 기존 지갑에 비해 검증 성능을 향상시키고 트랜잭션 크기를 축소하는 효과를 가지고 있다.

두 번째로, 외부 데이터베이스를 사용하지 않고 블록체인의 검색 성능을 향상시키기 위해, 블록체인 시스템 내부에 SQL 질의 연산을 통합하는 메커니즘을 제안한다. 이 메커니즘을 통해 이더리움 기반 블록체인 시스템 내부에 관계형 데이터베이스를 임베딩함으로써, 외부 데이터베이스나 별도의 사용자 정의 데이터 구조 없이도 스마트 계약 및 일반 트랜잭션에 대한 범위 질의를 실행할 수 있다.

마지막으로, 블록체인의 합의에 있어서 중앙화 문제를 해결하기 위해, 검증자를 상임위원과 운영위원으로 분리하고 이중 해싱과 검증 가능한 랜덤 함수(VRF)를 이용하여 무작위로 검증자를 선출하는 이중 위원회 증명 (PoDC) 합의 메커니즘을

제안한다. 이는 기존의 작업 증명 (PoW) 및 텐더민트와 같은 합의 알고리즘에서 발생하는 중앙 집중화 문제를 완화하고, 고정된 검증인 숫자를 통해 블록체인의 성능을 향상시킨다.

따라서, 본 논문은 블록체인 시스템에서 중앙 집중화 문제를 해결하고, 전반적인 보안과 성능 최적화를 달성하는 새로운 접근 방식을 제안한다.

Keywords: Blockchain, Multi-signature, T-ECDSA, Bloom Filter, Delay Function, Zero-knowledge Proof, Double Hashing, Verifiable Random Function (VRF)
Student Number: 2019-38471