IMPROVING THE SECURITY AND RELIABILITY OF APPLICATION SYSTEMS WITH BLOCKCHAIN TECHNOLOGY

ZHONGHAO LIU

MPhil

The Hong Kong Polytechnic University

2023

The Hong Kong Polytechnic University

Department of Computing

Improving the Security and Reliability of Application Systems with Blockchain Technology

Zhonghao Liu

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Philosophy

January 2023

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgment has been made in the text.

Signature: ____________

Name of Student: Zhonghao Liu

Abstract

As the backbone of cryptocurrencies, blockchain technology records data in a chain of blocks and brings features, including decentralization and immutability, by cooperating with other core technologies. Many researchers start to explore the combination between blockchain technology and application system for improving system security and reliability. In this thesis, we focus on two application systems of the electronic voting system (E-voting system) and audit log system, and improve their security and reliability by utilizing blockchain and related technologies.

Firstly, we note that existing E-voting systems cannot cover five core requirements in E-voting, i.e., auditability, privacy, authentication, correctness, and unreusability, which make them unpractical in the reality. We propose a Double Blockchain-based E-voting (DBE-voting) system, which consists of a private blockchain and a public blockchain. In the DBE-voting system, the voter information is only recorded in the private blockchain for further auditing and the voting results are recorded in both blockchains. This design ensures the voter's privacy can be protected in the private blockchain while the voting results can be queried in the public blockchains for verifying the correctness of the election process. Moreover, the ballot recorded in both blockchains is signed with a valid linkable ring signature to ensure authentication and unreusability. We propose an on-chain and off-chain hybrid storage mechanism to ensure the consistency and correctness of voting data in the private blockchain and public blockchain. To evaluate our system, we implement a prototype of our system by Hyperledger Fabric. Experimental results demonstrate that the throughput of our system can reach 29 transactions per second when the block size is 512 KB. Furthermore, the security analysis shows that DBE-voting is the first blockchain-based system that can meet all five requirements for E-voting simultaneously.

Secondly, we find that current audit log systems have a requirement of trusting the logger and auditor which may be compromised. Their centralized storage of log files also can cause single-point failure, preventing them from achieving data integrity. We propose a blockchain-based audit log system to address the above drawbacks while ensuring data integrity. We propose a general threat model in which the logger and auditor can both be untrusted and the log provider is trusted only when it generates log files. Under this threat model, we design a blockchain-based audit log system with multiple loggers and auditors to protect data integrity that can tolerate a certain number of malicious nodes. Our system adopts an efficient integrity proof generation method, which generates a sub-Non-Fungible Token (sub-NFT) for each log file and keeps it on the blockchain as the integrity proof. This method saves blockchain space and resolves the single-point failure problem by outsourcing log files to a distributed file system, the InterPlanetary File System (IPFS). To evaluate our system, we implement a prototype by Hyperledger Fabric. The results demonstrate that our system is reliable to tolerate one-third of colluded loggers and auditors. Our proof generation method can save approximately 50% storage space for Hyperledger Fabric compared with other blockchain-based audit log systems. Moreover, we provide security analysis to show that our system ensures log file data integrity under the general threat model.

Acknowledgments

Many people have given me valuable help in my thesis writing, including my supervisor, my colleagues, my parents, and my friends.

Firstly, I am deeply indebted to Prof. Bin Xiao, my supervisor, who guide me throughout the writing of the thesis. He took me through all the stages of writing this thesis. This thesis would not be in its current form without his consistently inspiring guidance.

Secondly, I would also like to thank my colleagues, Mr. Xinwei Zhang, Mr. Laphou Lao, Mr. Jiaping Yu, Mr. Rui Song, Mr. Yanjie Li, Prof. Shang Gao, and Mr. Zecheng Li who have offered me the great support for both my research and life at PolyU. Thank you for inspiring me during my research and helping me during my school life. It is my fortune and honor to work with such great colleagues and friends.

Finally, my thanks would go to my beloved family for their loving consideration and great confidence in me all through these years. I would also like to thank my other friends who have given me generous support and helpful advice over the past few years.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the boom in cryptocurrencies, blockchain technology as the backstop technology has attracted people's attention. Blockchain technology was first proposed and implemented with Bitcoin in 2008 by Satoshi Nakamoto. The data structure of the blockchain is an append-only chain structure that each block has a hash of the previous block and chain together. This data structure makes the block with packed data hard to modify or delete. Meanwhile, by comprising several other technologies, such as digital signature, cryptography, and distributed network consensus algorithm, blockchain can work in a decentralized environment and all the transactions that happened and data recorded in this decentralized manner can be immutable and auditable.

While blockchain technology is broadly used in cryptocurrencies, the researchers believe it can be implemented in diverse applications rather than just cryptocurrencies. Since the type of transaction data recorded in blockchain can be various, it can provide immutability, auditability, and many other properties for these different data. The blockchain is prepared to transform and improve the security and reliability of a wide range of applications, including supply chain and medical data management.

This thesis takes the electronic voting system (E-voting system) and audit log system as the research objects and we aim to improve the security and reliability of the E-voting system and audit log system with state-of-art blockchain technology and other related technologies.

1.1 Thesis Contribution

In this thesis, we propose several improvements in the security and reliability of the E-voting system and audit log system.

1.1.1 Privacy-preserving and Auditable blockchain-based E-voting System

Blockchain technology can construct a distributed and trusted ledger, which can be used for E-voting systems to ensure the security of voting data and improve government credibility. However, existing blockchain-based solutions cannot cover five core requirements in E-voting, i.e., auditability, privacy, authentication, correctness, and unreusability, which make them unpractical in the reality.

We propose a Double Blockchain-based E-voting (DBE-voting) system and this system contains two blockchains: private blockchain and public blockchain. Both two blockchains record ballot data, however, the voter's information is only recorded in the private blockchain and only for further auditing (auditability). The private blockchain does not reveal personal information to the public and it can protect the voter's privacy. The voting results can be queried in the public blockchain and the public can verify the correctness of the election process. Meanwhile, to restrict multiple voting (unreusability) and guarantee the voting right of valid voters (authentication), the votes are signed with valid linkable ring signatures before being recorded in two blockchains. For consisting the voting data recorded in two blockchains, we propose an on-chain and off-chain hybrid storage mechanism to ensure the consistency and correctness of these data. The security analysis shows that our privacy-preserving and auditable DBE-voting system that can meet all five requirements.

1.1.2 Secure and Reliable Audit Log System

Log files are widely used in digital forensics. It is important to ensure the data integrity of log files for auditing. However, current audit log systems have a requirement of trusting the logger and auditor which may be compromised. Their centralized storage of log files also can cause single-point failure, preventing them from achieving data integrity.

We propose a blockchain-based audit log system to ensure data integrity under a general threat model and resolve the concern of centralized storage. We build a general threat model which assumes the logger and auditor are untrusted and the log provider is trusted only when it generates log files. Under this threat model, we set multiple loggers and auditors in our system that can tolerate a certain number of malicious nodes while protecting data integrity. To handle the single-point failure and blockchain limited scalability problem, we propose an efficient integrity proof generation method. During the integrity proof generation process, our system generates a sub-Non-Fungible Token (sub-NFT) for each log file and the blockchain only stores small-sized sub-NFT as the integrity proof. The log files are outsourced to a distributed file system, the InterPlanetary File System (IPFS), to resolve the single-point failure problem. Through these designs, our audit log system can be more secure and reliable, save blockchain storage space, and resolve the single-point failure problem.

1.2 Thesis Outline

The remainder of this thesis is structured as follows. Chapter 2 introduces the main technologies we used in this thesis. Chapter 3 presents our work on DBE-voting. The work of our blockchain-based audit log system is proposed in Chapter 4. Finally, we conclude the thesis and give our suggestions for future research in Chapter 5.

Chapter 2

Background

In this chapter, we take an overview of the main technologies used in our thesis. We first introduce Hyperledger Fabric, an advanced blockchain technology that is utilized in both two application systems. Then, we discuss the cryptography applied in the DBE-voting system. Finally, we introduce other technologies that are used in our audit log system.

2.1 Blockchain and Hyperledger Fabric

The structure of blockchain is an append-only data structure. As shown in Fig. 2.1, the transactions and data are packed in a block, and the block links the previous block by solving the mining puzzle and storing the previous block hash value. If a new block wants to link to the blockchain, it must reach a consensus in a distributed network that nodes in the network approve the block and append it to their local blockchain. This design guarantees immutability. The type of blockchain includes public blockchain, private blockchain, and consortium blockchain. Public blockchains are widely used in cryptocurrency, for example, Bitcoin [48] and Ethereum [64]. They reserve read and write access to any user who joined the

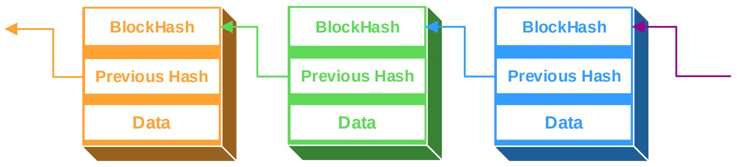


Figure 2.1: The blockchain data structure.

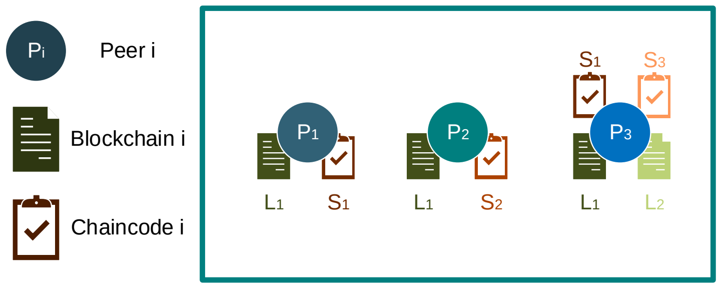


Figure 2.2: An example of Hyperledger Fabric network. This figure shows some characteristics in a Hyperledger Fabric network that peer node in one network can contain multiple blockchians and install different chaincodes.

network. Private blockchains are the opposite of public blockchains. They only allow authorized participants to hold read and write access. Consortium blockchains may leave the read access to the public. They select a set of nodes to operate the consensus process [23] and allow these participants to hold write access. The node topology is decided by the consensus policy and can be changed if the consensus policy updates.

We use Hyperledger Fabric, a consortium blockchain, to build our E-voting system and audit log system. As shown in Fig. 2.2, Hyperledger Fabric allows different peer nodes to operate the same blockchain with different chaincodes or operate different blockchains with their related chaincodes in one system. It designs a novel consensus mechanism to ensure the correctness of executing a chaincode.

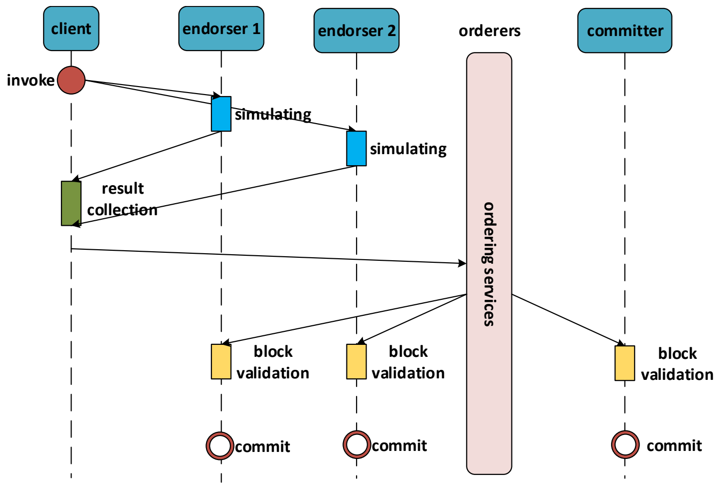


Figure 2.3: Execute-order-validate transaction process. This is the main process by which Hyperledger Fabric reaches consensus.

Meanwhile, compared with the public blockchain platform (e.g., Bitcoin, Ethereum), Hyperledger Fabric is better suited to serving large-scale modern systems since it can achieve higher throughput and lower confirmation latency [6]. These properties improves the system's security, protects users' privacy and supports the design of our systems.

We introduce the key components shown in Fig. 2.2 and the consensus mechanism in Hyperledger Fabric.

Peer: Peers are the foundation of the entire blockchain network. It is the customized operation node in the Hyperledger Fabric system. It stores and updates the blockchain locally, and it can install chaincode to execute transactions. The peer calls the endorser if it can execute the transaction, or it calls the committer if it only stores and updates the blockchain. In addition. all peers can validate the transactions and update the blockchain.

Orderer: Different from peer nodes, the orderers are not responsible for executing and validating the transaction. They only focus on receiving the transaction, generating block, and sending it to peer nodes.

Chaincode: The smart contract installed in peers is also called chaincode in Hyperledger Fabric. As the smart contract, the chaincode defines the logic and function of a specific network, and the client creates the transaction by invoking a function in the chaincode. However, the chaincode only be executed based on the designed endorsement policy, which may not be a decentralized executing environment and could be updated.

Channel: Hyperledger Fabric allows multiple blockchains to exist in one system, and these blockchains are all connected to the same ordering service. Each such blockchain is called a channel and may have different peers as its members.

Endorsement policy: It is a boolean expression to guide peers on how to determine whether the transaction is approved or not. It is defined in the endorsers when instantiating the chaincode in endorsers. It could be changed and updated during the system process.

Execute-order-validate architecture: The Hyperledger Fabric uses execute-order-validate architecture to help the system reach the consensus. In Fig. 2.3, the consensus mechanism can summarize into three phases: executing phase, the ordering phase, and the validation phase. In executing phase, a client sends a transaction to endorsers specified by the endorsement policy. The endorsers simulate the transaction and return the simulation result to the client. After execution and transactions enter the ordering phase. In this phase, all transactions are totally ordered sequence, packed in a block, and the block is broadcast to all peers. Finally, each peer validates the changes caused by the new block and the consistency of the execution in the validation phase.

Fabric Software Development Kit: To address some of the original requirements that customers face directly in blockchain networks, the Hyperledger Fabric design the Software Development Kit (SDK) to support the developers with powerful and convenient API.

2.2 Linkable Ring Signature

This technology is not directly related to Hyperledger Fabric, but it is an essential component in our E-voting system for helping to satisfy some requirements. Unlike traditional public key encryption, which could only encrypt the content, the ring signature can hide the origin and destination by increasing the single public key to a set of keys. Moreover, the linkable ring signature applies the linkability in the ring signature by implementing the one-time keys or other methods in the public key ring. The linkability could limit the same real public key sign the transaction once, and even it hides in the public key vector. This property not only protects the users' privacy but enhances system security. For example, Monero [50], a strongly decentralized anonymous cryptocurrency, uses the linkable ring signature to prevent the double-spending attack.

2.3 InterPlanetary File System (IPFS)

The IPFS is a distributed file system where peers are connecting to each other to store files [9]. IPFS synthesizes successful design principles from previous peer-to-peer systems, including Distributed Hash Tables (DHTs), BitTorrent, Git and Self-Certified File Systems (SFS). It provides a new platform for distributing and storing large data.

IPFS nodes store IPFS objects locally and connect and transfer objects. These objects represent files (blob) or other structures (list, tree, commit). Files may be separated into several objects if they are too large. On top of these objects, IPFS builds Merkel DAGs, a directed acyclic graph, to link these objects. Merkel DAGs bring one of the most important properties is content addressing. All contents are uniquely identified by their multihash checksum which is also called content identification (CID). Users can get a file that has been published on IPFS by inputting the related CID to any IPFS node. All content is verified with checksum. If data is tampered with or corrupted, IPFS can detect it and the user can perceive it.

In our audit log system, we outsource log files to IPFS. Its distributed storage mechanism helps us release the concern of single-point failure. The CID generated by IPFS can be treated as integrity proof, stored on the blockchain, and verified by our system's auditor.

2.4 Non-Fungible Token (NFT)

NFT is a cryptocurrency [26] that can be defined by Ethereum's smart contracts [64]. According to Ethereum's token standard, we can distinguish each token with an identifiable sign. NFT can bond with virtual or digital properties as their unique identifications, which makes it impossible to trade equivalently like standard coins such as Bitcoin [48]. Specifically, creators can apply NFTs to smart contracts to prove ownership of digital assets such as audio, video, images, artworks [28], etc., and provide exchange opportunities

In this thesis, we extract the property of NFT in Ethereum and realize it in our audit log system called sub-NFT. In the current stage, sub-NFT is stored in Hyperledger Fabric, it can prove the existence of one specific object and we use this property as a log file integrity proof.

Chapter 3

DBE-voting: A Privacy-preserving and Auditable Blockchain-based E-voting System

3.1 Introduction

Voting is the embodiment of justice and democracy, and a tool for citizens to use their rights legally. In a traditional voting scheme, the entire election is processed by humans, which is resource-intensive and prone to human error [33]. Some governments have replaced traditional paper voting with electronic voting (E-voting) in recent years to make the voting process more efficient. Voters can register and vote through their electronic devices, and the records are received by the government server and saved in the database. The civil servants can launch the computer program to traverse the database and generate the final result. However, the traditional E-voting scheme with security problems, such as data tampering and privacy leaking, leads many governments to reject this method [29].

Since blockchain technology can provide a distributed and immutable ledger to store data, this technology has been widely used in many fields, e.g., healthcare [8], Internet of Things [63] and supply chain [56]. Recently, the blockchain has also been introduced to E-voting system for addressing the security problems in traditional E-voting systems, which called blockchain-based E-voting (BE-voting) [35]. In the BE-voting systems, the electronic ballots are recorded in the immutable ledger and distributed across the blockchain network to improve the data authentication, and voters' identities are encrypted by cryptography when they make a transaction with the BE-voting system to protect voters' privacy. Without loss of generality, for a reliable BE-voting system, it should satisfy the following core requirements:

1) *Auditability*: Only registered and verified voters can vote, and the records should keep who are involved in the voting;

2) *Privacy*: The voter's personal information and the voting process will be kept in private;

3) *Authentication*: The ballot must be generated by valid voters and nobody can change them;

4) *Correctness*: The recording and counting processes should execute correctly;

5) *Unreusability*: A valid voter can only vote once;

Previous approaches can meet some requirements. For the privacy, most existing work encrypts the voters' identity [32, 35, 44, 49, 61, 67] or customizes the system framework [36, 39]. They implement the digital signature technique (e.g., blind signature and ring signature) to improve the authentication and enhance voters' privacy. For the correctness of the recording and counting process, some approaches program these processes in the smart contract [20, 35, 36, 49, 61], and others design the voting protocol [31, 39]. For the unreusability, a small number of researchers apply the E-wallet [35] or ring signature [61] or design algorithms [32, 65] or zero-knowledge authentication [49] to restrict voters from voting multiple times. For the auditability, only a few studies attempt to achieve it through utilizing the smart contract [20] or customizing the system framework [65] to involve voters' identities in the records. However, the existing BE-voting systems do not meet all the requirements, which remains a concern for the voters and delays the adoption of E-voting.

Meeting all five requirements in a BE-voting system is still a huge challenge. For example, if ballot recorded in a single blockchain contains voter personal information for auditing purposes and is kept confidential for privacy protection, voters and public cannot verify the correctness of election process since the data are encrypted and they are not allowed to access the system. Similarly, if the system satisfies the privacy and correctness, the auditability may not be satisfied. The single blockchain, which records voting data without voter personal information, can be accessed by anyone to verify the correctness. However, without the related voter personal information, the system cannot prevent the impersonating attack [4], audit the election result, and investigate the legal issues.

In this chapter, we propose a Double Blockchain-based E-voting (DBE-voting) system, including a private blockchain and a public blockchain to meet all five requirements. Both two blockchains record the same voting data. The private blockchain also records voters' encrypted public key, voting location, and device parameters for further auditing, and no one will be able to access the private blockchain until the election is finished for privacy protection. The public blockchain allows voters to query their ballots and supervise the election tallying process to verify the correctness of the election. Besides, the linkable ring signature [58] is used to authenticate and encrypt the voters' identification and restrict voters from voting multiple times.

Meanwhile, cooperating with two isolated blockchains brings another challenge, which is maintaining the data consistency between two blockchains, because of the low scalability [10] and lack of interoperability [45] in the blockchain network. We cannot require the voter to revote if the transaction only fails in the public blockchain since the valid data recorded in the private blockchain cannot be reversed.

We propose an on-chain and off-chain hybrid storage mechanism to overcome this challenge. The off-chain database is set up to store the public blockchain transaction data and server can retrieve it to remake the transaction only with the public blockchain if the transaction fails in it. Transaction delay between two blockchains is allowed by comparing the block height and timestamps in these two blockchains.

In general, we present a privacy-preserving and auditable DBE-voting system that can meet all security requirements. The main contributions in this chapter can be summarized as follows:

(1) **Double Blockchain-based E-voting System.**We propose a novel DBE-voting system, which consists of a private blockchain and a public blockchain, to meet all the requirements for a reliable E-voting system.

(2) **Cross-chain Data Consistency.** We propose an on-chain and off-chain hybrid storage mechanism, which ensures data consistency between the private blockchain and public blockchain.

(3) **System Implementation and Security Analysis.** We develop a prototype based on Hyperledger Fabric [6] and evaluate its performance. The results demonstrate that our system has a good performance when the block size is 512 KB, and the data consistency method has not become the bottleneck at the current stage. The security analysis proves that our system can satisfy all the requirements.

The remainder of this chapter is structured as follows. Section 3.2 provides a literature review of the existing BE-voting research. The system model and the data consistency method is described in section 3.3 and section 3.4. Section 3.5

Table 3.1: Comparison between related research.

Relevant Literature	Auditability	Privacy	Authentication	Correctness	Unreusability	Implementation
Ethereum-based E-voting system [35]	×	√	√	√	√	×
Permissioned BE-voting system [31]	×	×	×	√	×	√
zVote [49]	×	√	√	√	√	√
Decentralized BE-voting protocol [32]	×	√	√	×	√	√
Large-scale BE-voting protocol [61]	×	√	√	√	√	√
Prêt à Voter E-voting method [39]	×	√	√	√	×	√
Permissionless BE-voting protocol [44]	×	√	√	×	×	×
BroncoVote [20]	√	×	√	√	×	√
BE-voting system in P2P network [65]	√	×	√	×	√	×
Privacy-preserving BE-voting protocol [67]	×	√	√	×	×	×
Decentralized BE-voting system [36]	×	√	√	√	×	×
Our System (DBE-voting)	√	√	√	√	√	√

reports the evaluation results and analyses the requirements. Finally, we conclude this paper in Section 3.6.

3.2 Related work

In literature, there exist some research that try to realize a reliable E-voting system. We first introduce some conventional E-voting system. Then, we introduce some blockchain-based E-voting system based on the number of satisfying requirements.

3.2.1 Conventional E-voting researches

In 1981, Chaum [15] firstly proposed using the public key cryptography to encrypt the ballot, which allowed the voters voting anonymously. After that, many countries have used various E-voting systems since 21st century. In 2000, the United States became the first country to use the E-voting for a political election [25]. Between 2002 and 2007, the number of countries using E-voting system increased, including the United Kingdom (2002), Estonia (2005) and Canada (2006) [25,30,46]. In 2008, Ben Adida developed the first web-based, open-audit voting system, Helios [4,], which lays the foundation for the future E-voting applications. Helios, in 2009, held a presidential election in Belgium's Université Catholique de Louvain [40]. Based on the Helios's innovations, the United Stated company Voatz [2] proposed the first blockchain-based E-voting application in 2014. And, in 2018, the Voatz made a pilot project in West Virginia, allowing the voters to vote from oversea in the midterm election [40].

3.2.2 Blockchain-based E-voting researches

Ngyuen et al. [49] propose a BE-voting that satisfied most of the requirements for a reliable system, but it did not meet all of them. The authors wrote the smart contract in the system to maintain the correctness of the voting process. They designed a registration process to verify voters' identities, and the non-interactive zero-knowledge proof was used to protect the voters' privacy. They utilized zero-knowledge authentication to restrict voters' voting multiple times. However, they did not provide the auditability that the records did not keep who were involved in them and did not mention what type of blockchain to realize and evaluate the entire system. Wang et al. [61] proposed a blockchain-based voting protocol for large-scale voting, and this design met most of the requirements. They designed the registration process to verify the voters' identities and applied the ring signature to ensure authentication, unreusability and to protect voters' privacy. They utilized the smart contract and public verification mechanism to ensure the correctness of the recording and tallying process. However, even if they launched experiments to evaluate the performance of the protocol, the system architecture in this research was unknown, and the audit process was unclear.

These two are the most comprehensive work in the existing research since they satisfy the most requirements except the auditability, and the detail of comparison is shown in Table 3.1.

According to Table 3.1, only a little work satisfy the auditability. Yi [65] proposed technology to employ blockchain in the E-voting system. He designed three models in his system to meet the requirements. The synchronized model utilized the distributed ledger technology to avoid the forgery of votes and store the voter ID with the corresponding ballot to provide the auditability. The user credential model applied the elliptic curve cryptography to ensure authentication. The withdrawal model allowed voters to replace their votes and accept the latest vote. However, the voters' privacy was a problem since the voter ID could be exposed to the adversary, and he did not provide the correctness. Dagher et al. [20] programmed three Ethereum smart contracts to satisfy the requirements. The registrar contract verified the voter identity and ensured the authentication. The creator contract and voting contract provided the auditability and correctness. However, the voting contract stored the voter's Ethereum address in the record and threatened the voter's privacy. Since Ethereum is a public blockchain network, voters' account and their transactions were directly exposed to the public during the election, and they did not provide the unreusability.

As shown in the above work and the Table 3.1, we conclude that the existing research do not provide a comprehensive BE-voting system to satisfy all the abovementioned requirements. Some proposed designs only consider a couple of requirements and implement them in the E-voting system. Some work only introduce the E-voting system's requirements and propose the design, but they do not implement any E-voting system. The DBE-voting system we proposed satisfies all the requirements mentioned above, and we implement a prototype to test its performance.

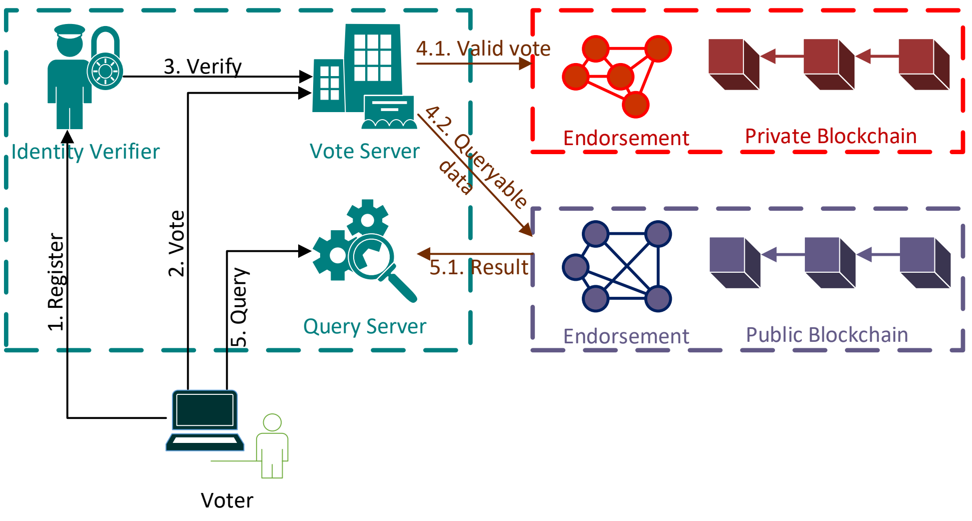


Figure 3.1: The proposed DBE-voting architecture. This figure shows the main structure of the entire system.

3.3 The Proposed DBE-voting

This section will devise a double blockchain-based electronic voting system, which satisfies and optimizes identified requirements and considerations. In the following subsection, we place the roles and components for implementing an E-voting system. Then, we describe the entire voting process and elaborate on the data process in each phrase.

3.3.1 System Overview

The system can divide into five different parts, and each one of them is responsible for the specific task in one election. We start by describing each system part, followed by the election process. As shown in Fig. 3.1, some system parts have more than one character, and we unite them together. Each part of the proposed system is indispensable and relatively independent.

Identity Verifier, Vote Server, and Query Server: Verifying the voter's identity, launching the election, receiving the vote from the valid voter, sending the vote tho the blockchain, querying the vote stored in the blockchain, and announcing the result.

This system part manages the lifecycle of the election. Identity verifier registers and verifies the voters' identities, which satisfies our E-voting system's first part of auditability. It supplies the API for the valid voter to generate the linkable ring signature locally and protect their privacy. The vote server verifies the validity and linkability of each voter's ring signature to satisfy the authentication and unreusability in our system. To ensure the correctness, the recording and counting process are processed by private and decentralized blockchain network. Moreover, the vote server generates the receipt for each voter to prove the correctness of the recording process. Query server allows voters to query their ballots through the receipt and supervise the tallying process to verify the correctness of the recording process and counting process.

Voters: For elections to which they are applicable, voters first need to authenticate themselves through the identity verifier. Then, they can cast their vote through the government vote server and get the exclusive receipt generated from the vote server. Moreover, they can send the receipt to the government query server to search their ballot, and they can verify their vote in their local device after they get the ballot from the query server.

Private Blockchain: Invoking the private chaincode, reaching a consensus through the endorsement policy, and saving the valid vote and voter's personal information in the private blockchain. The private blockchain receives valid votes from the vote server and processes them independently. The private chaincode and endorsement policy ensure the correctness of the recording process in the private blockchain. The voters' unique public key ring, voting location, and device parameter are recorded in the private blockchain to satisfy the second part of auditability in the system. No one can access the records in the private blockchain until the election is over to protect the voters' privacy.

Off-chain Database: Saving the queryable data before the vote server invokes the public chaincode, helping the vote server maintain the data consistency between the private blockchain and public blockchain. In addition to recording the queryable data, the off-chain database records the creation time when each valid vote save in the private blockchain.

Public Blockchain: Invoking the public chaincode, reaching a consensus through the endorsement policy, and saving the queryable data in the public blockchain. The voter can use their receipt to query the data from the public blockchain based on the query server platform. After an election, the vote server launches the public chaincode to tally the ballots recorded on the public blockchain. The public chaincode and endorsement policy ensure the correctness of the recording process and tallying process in the public blockchain. Voters can query their ballots and supervise the tallying process on the public blockchain to verify the correctness of the recording process and counting process.

3.3.2 Election Process

In our work, each election process is cooperated by five system parts (shown in Fig. 3.1) that receive the valid data from upstream, process it, and send it to the downstream. The followings are the main activities in the election process:

Election creation

Election administrators design and write the election-related options in the vote server platform, for example, the candidate list and the voting district list. After that, they design the private chaincode and public chaincode with the decided option lists. Then, they install the chaincodes in the corresponding organizations, which affiliate with either private blockchain or public blockchain, and instantiate these organizations to create the first block of the election. At the same time, the administrators create the table of the off-chain database and design the query server platform, which displays the query result and data in the database. Finally, they build the connection using the fabric SDK between the central servers and blockchains.

Voter registration

The identity verifier processes the registration of a voter. It currently uses offline registration and online verification to authenticate the voters' identities. The offline registration requires the voter to send their personal information, including the ID number and phone number, to create an account on the identity verifier platform. The online verification requires the voter to send their real-time face ID and fingerprints to the identity verifier. The staff behind the identity verifier verify the voter's personal information and real-time biological information. If the voters pass the verification, they can generate the private key and the public key ring through the API supplied by the identity verifier in their local devices. The voting server checks for the validity and linkability of the signature when voters send their signed vote to the vote server.

Vote verification

As shown in the Algorithm 1, after the valid voter fulfills the vote and signs it, the vote server verifies the validity of the signature. Then, it checks for the linkability with signatures that have been recorded in the private blockchain. The truth of signature validity represents the voter has a valid identity, and the truth of signature linkability represents the voter may have voted. The vote server firstly checks the signature validity. Then, it checks the signature linkability if the validity is true.

In the final, only one condition can make the system move to the following process: The validity is true, and the linkability is false.

Algorithm 1 Verify the validity and linkability of signed ballot

```
1: function VERIFY($Ring$, $Sign$)
2:     $H \leftarrow GetHistoryForKey("Signature")$
3:     if $Validity(Ring, Sign) \neq true$ then
4:         $return\ false$
5:     end if
6:     for $each\ s \in H$ do
7:         if $Link(Sign, s) = true$ then
8:             $return\ false$
9:         else
10:            $continue$
11:        end if
12:    end for
13:    $return\ true$
14: end function
```

Vote transaction in private blockchain

When the signed vote passes the verification, the vote server makes a transaction with the private blockchain through the fabric SDK (shown in Fig. 3.2). Since the blockchain network is isolated and its process is independent from the vote server, the fabric SDK can be treated as "gateway" to build a connection between the vote server and private blockchain network. After the connection is built, it sends the transaction proposal to the endorsers by invoking the functions in fabric SDK. The transaction proposal includes the operation type, the function name, public key ring, candidate name, voting district, the linkable ring signature, and other parameters. The consensus mechanism in Hyperledger Fabric initiates when the endorser peer

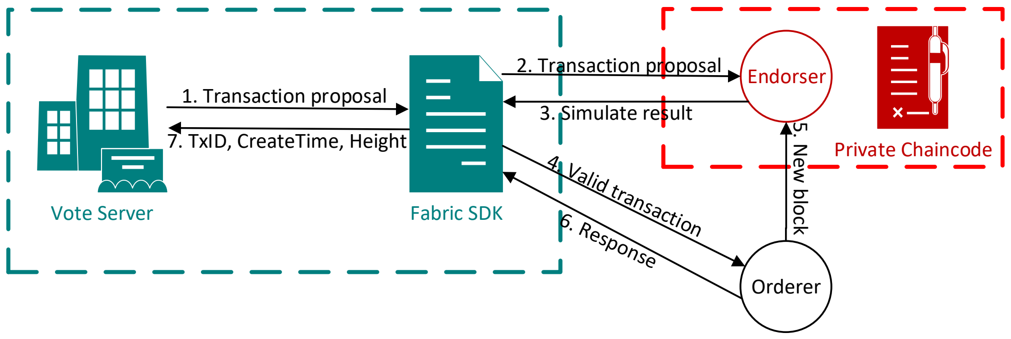


Figure 3.2: Transaction flow of signed ballot in the private blockchain. This figure shows the detail of transaction process in Fig. 3.1. The fabric SDK can be viewed as a "gateway" file installed in the vote server to help the vote server transact with the private blockchain. The endorser and orderer are nodes in Hyperledger Fabric. Endorser processes the transaction by invoking the chaincode, and orderer generates new block to store it.

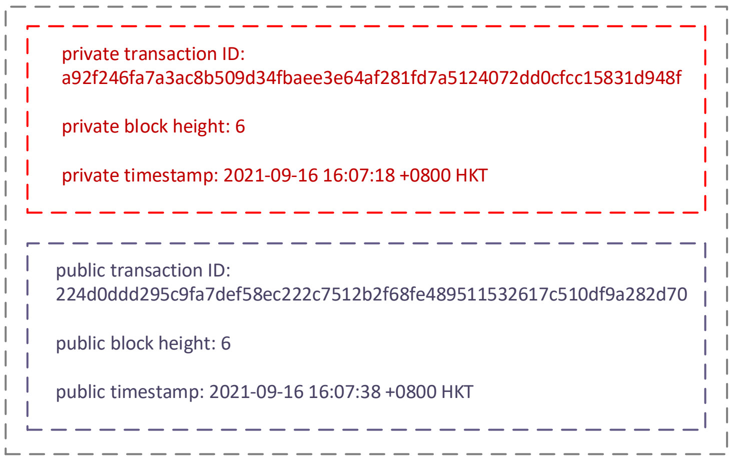
Table 3.2: Example of transaction in public blockchain and record in off-chain database. (The TxID, height and create time are generated by the private blockchain)

TxID	Height	Candidate	Location	Signature	Create time
8dfa76...	4	Jason	Haidian	616242...	16:07:18

executes the transaction proposal. Supposing the result of the transaction proposal reaches a consensus in the private blockchain. In that case, the orderer returns the transaction's status, including transaction ID, block height and create time, back to the vote server. Table 3.2 shows the combination of proposed parameters and the response, which is the new transaction format in the following system processes.

Data store in off-chain database

The off-chain database stores the combined data (shown in Table 3.2) in the off-chain database before the vote server makes a transaction with the public blockchain. En-



private transaction ID:
a92f246fa7a3ac8b509d34fbaee3e64af281fd7a5124072dd0cfcc15831d948f

private block height: 6

private timestamp: 2021-09-16 16:07:18 +0800 HKT

public transaction ID:
224d0ddd295c9fa7def58ec222c7512b2f68fe489511532617c510df9a282d70

public block height: 6

public timestamp: 2021-09-16 16:07:38 +0800 HKT

Figure 3.3: The receipt that voter received when the ballot was recorded in blockchains successfully.

suring that the data queried by voters from the public blockchain have corresponding records in the private blockchain, the system must maintain data consistency between the private blockchain and the public blockchain. So, before the vote server transactions with the public blockchain, the system stores the queryable data in the off-chain database. We use the off-chain database, and the details of data consistency process are introduced in the next section.

Vote transaction in public blockchain

After the system stores the cross-chain data in the off-chain database, the vote server transactions with the public blockchain through the fabric SDK. The process details are similar to the vote transaction in the private blockchain. When the vote server receives the result returned from the public blockchain, it generates the unique receipt and sends it back to the voter. The content of the receipt, as shown in Fig. 3.3, is combined into two parts, and each part contains the response data from the blockchain. Voters can use the receipt generated from the vote server to their vote recorded in the blockchain.

Ballot query

Voters who receive the receipt from the vote server successfully can access the query server to search their ballots recorded in the blockchain. The voter needs to input the public transaction ID, as the keyword, to the query server platform. The query server makes a query operation with public blockchain through the fabric SDK, which process is similar to the vote transaction in the private blockchain. Table 3.2 shows the search result returned from the query server, and the transaction ID represents the transaction made in the private blockchain. Voters can compare the unique private transaction ID recorder in the receipt with the result shown in the query platform and verify the validity of the linkable ring signature in their local device.

Tallying result

When the election is over, the vote server starts tallying the results based on the ballots recorded in the public blockchain. The vote server invokes the tallying function programmed in the chaincode, and endorsers of public blockchain traversal the entire public blockchain and select the vote transaction recorded in it. Then, They count the ballot number for each candidate, compare them to get the result, and make a consensus. Eventually, the vote server gets the consistent result and publishes the final result to the citizen. During this process, the citizen can supervise the entire tallying process. If the final result is correct and the audit process is complete, the records in each peer's local storage need to be eliminated before the next election to ensure the voter privacy.

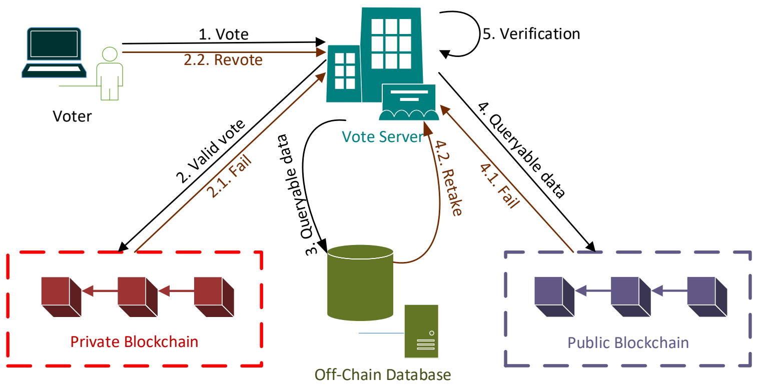


Figure 3.4: The data consistency process in the proposed system. This figure shows the detail of data consistency process in Fig. 2. The vote server requires the voter to revote if the transaction fails in the private blockchain. It stores the queryable data in the controlled off-chain database before transacting with the public blockchain. The queryable data is recaptured by the vote server, and if the transaction fails, it is recreated in the public blockchain. Finally, the vote server verifies the data consistency between the private blockchain and the public blockchain.

3.4 Data Consistency in DBE-voting

To ensure the ballot information searched by voters and counted in the tallying process does have the corresponding valid ballots recorded in the private blockchain. As shown in Fig. 3.4, Our proposed system returns the errors to voters' local devices if the transaction fails in the private or public blockchain. Then, the vote server remakes the transaction again. After the ballot is recorded in the public blockchain, the voter server invokes the specific verify function to check the consistency of the transaction above.

3.4.1 Ensuring the transactions are recorded in private blockchain

The first step to realize the data consistency is recording the ballot in the private blockchain successfully. According to the consensus mechanism in Hyperledger

Fabric (shown in Fig. 2.3), if the transaction fails in executing phase, it will not be packed in the new block; If the transaction fails in validation, the packed transaction will be marked as invalid.

The vote server sends the transaction proposal to different endorsers through the fabric SDK and waits for simulated results. If the simulated results that the vote server received are different, the transaction failed and the vote server will notify the voters and asks them to vote again. Another situation is that the parameters in the transaction proposal do not satisfy the conditions set in the private chaincode. The endorsers return an error to fabric SDK even it satisfies the endorsement policy. Supposing the transaction proposal passes the conditions specified in the private chaincode and the simulated results are same. In that case, this transaction will be sent to the orderer to be packaged into a new block.

The orderer sorts the received transactions by arriving time, generates the new block by calling a timeout or reaching the maximum transaction number in one block, and sends the block to the peer nodes. After the peer nodes receive the block sent from the orderer, they evaluate these transactions based on the default system chaincodes and the endorsement policy. If it does not pass the evaluation, the transaction will be marked as an invalid transaction. Finally, the peer nodes update this new block to their local ledger and return the result to the vote server. If the transaction success, the result contains the transaction ID and other related data stored in the peer's local ledger. Otherwise, the vote server notifies the voters and asks them to vote again since their ballots are marked as invalid transactions in the block.

This process ensures the ballot is recorded in the private blockchain when the vote server receives the transaction ID, create time, and block height from Hyperledger Fabric.

3.4.2 Ensuring the transactions are recorded in public blockchain

The second step to realize the data consistency is guaranteeing the queryable data (shown in Table 3.2) are recorded in the public blockchain. The vote server stores the queryable data in an off-chain database before it transactions with the public blockchain. The transaction flow in the public blockchain is similar to the previous step, but it is different when they handle invalid transactions.

In the previous step, if the vote server receives an error or the transaction is invalid, the vote server can return an error to the voter's local device and ask the voter to vote again. However, this method is ineffective when the vote server transactions with the public blockchain. The valid vote has been recorded in the private blockchain, and the voter cannot vote again, based on the requirement of unreusability. So, the vote server can only return the error to the voter's local device and solve the error by itself.

We designed the method to set the off-chain database to store the queryable data before the transaction starts. If the vote server fails to make the transaction in the public blockchain, it can retake the queryable data from the off-chain database and make a transaction again. In the future, developing more methods to treat and prevent transaction failure is one of our main tasks because requiring the vote server to relaunch the transaction is a necessary process but not a real solution when facing transaction failure.

3.4.3 Verifying the data consistency before generating the receipt

The last step to data consistency is verifying the correlation of properties that the vote server gets from the private and public blockchain. As aforementioned in vote transactions, the vote server gets the response, including the transaction

ID, block height, create time, and other properties, when the transaction succeeds. These properties represent the position and status of the transaction recorded in the blockchain. We can use these fixed properties and the system setting parameters to build a special relationship, verifying the data consistency of one transaction recorded in the private and public blockchain separately.

$$(T_{pub} - T_{pri}) <= (H_{pub} - H_{pri} + 1) * BatchTimeout \tag{3.1}$$

As shown in Equation 3.1, the *BatchTimeout* means the duration of generating one block, the T indicates the create time, and the H means the block height. The *BatchTimeout* is set at the system design and fixed when the system is launched. The left side represents the actual duration of recording the queryable data into the public blockchain. The right side represents the maximum theoretical duration of recording the queryable data into the public blockchain.

This logic is adaptive for verifying the data consistency in the situation that the vote server meets the error when making the transaction with the public blockchain. For example, a particular transaction fails to store in the public blockchain, and the vote server has to spend extra time to make the transaction again until the transaction is recorded in the public blockchain successfully. The public blockchain might generate new blocks in this period if the error does not affect the system's operation. So, this transaction's block height in the public blockchain may be larger than that in the private blockchain.

In this verification process, we assume the public blockchain records other transactions successfully, even the error happened in a particular transaction. This condition ensures the Equation 3.1 working correctly. In the future, we need to consider the error which might stop the entire operation in the public blockchain because this type of error can stop public blockchain from generating new blocks and making the Equation 3.1 ineffective. Meanwhile, we need to enhance the data security in the off-chain database to prevent data tampering and other attacks.

3.5 Evaluation

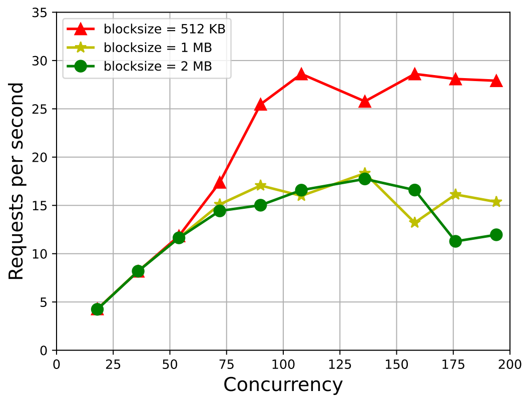
This section reports some preliminary performance numbers of our E-voting system, even though it is not performance-tuned and optimized. Then, we analyze whether the system meets the requirements summarized in section 3.1.

The Hyperledger Fabric implements our E-voting system. According to the [6], we know that the Hyperledger Fabric is a complex distributed system; its performance is related to many parameters, including the parameters of implementing a system, the network parameters, the hardware on which the nodes run, and others. In our experiment, we set different parameters to observe our system performance changes. Moreover, we design the off-chain database and related verifying algorithm to ensure the data consistency between the private and public blockchain. This implementation might impact the system performance except for the influence from Hyperledger Fabric, so we decide to remove this implementation and observe the change of our system performance. Other in-depth performance evaluations of our system are postponed to future work.

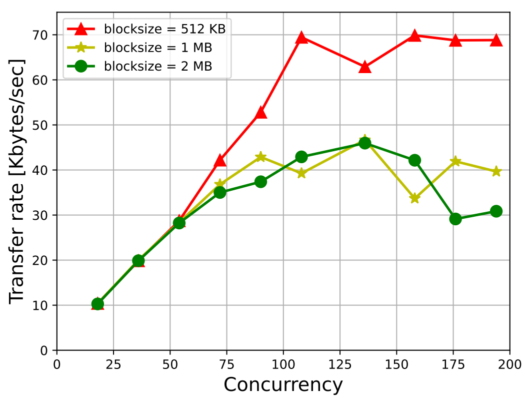
3.5.1 Experiments

Setup

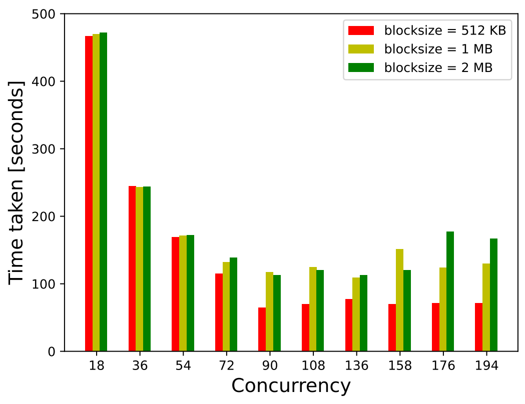
We implement our system based on the Hyperledger Fabric [6]. In our experiment: (1) the prototype is realized on Fabric v1.4.2-preview and the system performance is monitored by a local logging process, (2) peers are hosted separately in Docker containers [12] as dedicated VMs, (3) all clients are hosted in one node, (4) a single Fabric orderer node offers ordering service, (5) each blockchain has 3 endorsers and



(a)



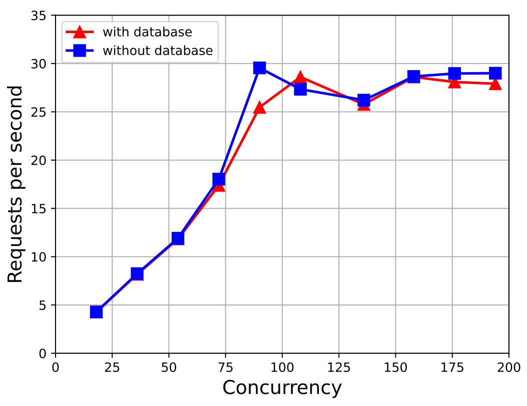
(b)



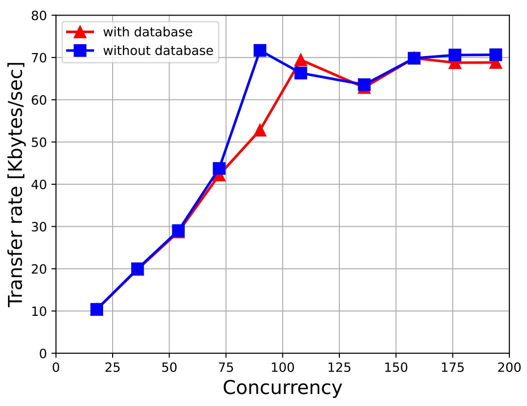
(c)

Figure 3.5: The impact of block size parameter on system performance. (a) The impact of block size on throughput. (b) The impact of block size on transfer rate. (c) The impact of block size on time taken for test.

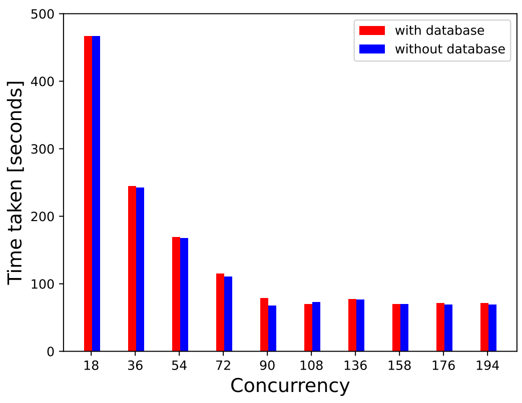
each endorser represents one organization (Org), (6) the endorsement policy is designed to satisfy the majority rule, which is implemented as *OR(AND(PriOrgOne, PriOrgTwo), AND(PriOrgOne, PriOrgThree), AND(PriOrgTwo, PriOrgThree))*, and (7) the sqlite3 is applied as the off-chain database to store the cross-chain data.



(a)



(b)



(c)

Figure 3.6: The impact of data consistency process on system performance. (a) The impact of data consistency process on throughput. (b) The impact of consistency process on transfer rate. (c) The impact of consistency process on time taken for test.

Methodology

In every experiment, we assume all voters have passed the identity verification and ignore the time that voters fill the ballot form. They send the request to the vote server and wait to receive the receipt. We use the Apache Benchmark as the benchmarking tool to test the performance of our E-voting system. The total number of requests is not changed. We regularly increase the concurrency number in each test to see system performance changes. In addition, the *BatchTimeout*, as a system parameter in Hyperledger Fabric, can be treated as the response time when the voters get their receipt, and we set this parameter as 2 seconds for a good user experience.

Experiment 1: Impact of block size

According to the [6], we understand that the block size is a critical system parameter to impact the throughput. We set the maximum number of messages to 800 to eliminate the impact of unrelated parameters and compare the system performance at the block sizes equal to 512 KB, 1 MB, and 2 MB. The results are depicted in Fig. 3.5.

Fig. 3.5(a) and Fig. 3.5(b) show that when the concurrency number is larger than the 72, the system performance of 512 KB is better than those of 1 MB or 2 MB. Moreover, the 512 KB block size system is more stable than others when the concurrency number is larger than 90. The performances of 1 MB and 2 MB are not stable and stop increasing because the nodes crash when the concurrency number is larger than 136. In future work, to make the system more stable, we can apply the crash-fault tolerance method in the system and increase the number of endorser nodes to reduce the computing load of each node.

The Fig. 3.5(c) shows the time consumed in each test. We can observe that as the number of concurrencies increases and the total number of requests stays the same, the time taken for the test decreases. Since the system performance of 512 KB is better than those of 1 MB or 2 MB, its time taken is generally less than others.

Although the system throughput in this experiment is not sufficient for real-world government elections, we demonstrate the system's feasibility. To satisfy all the abovementioned requirements, we implement two relatively isolated blockchains in one system, and our experiment shows that the design is achievable. Due to equipment limitations, the system prototype is implemented on a personal computer on which all peer nodes are hosted. We believe our system can achieve better performance when each node can be implemented on a single server.

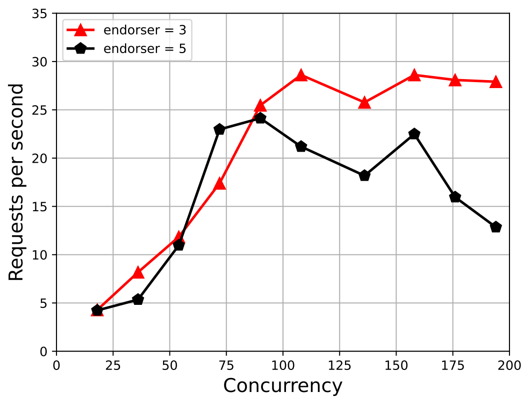
In this experiment, we also observe the size of the vote transaction and queryable data transaction from the local Docker log. In particular, the 512KB-size block can contain 72 vote or 71 queryable data transactions, so the average transaction size is 7.11 KB for the vote transaction and 7.21 KB for the queryable data transaction. The transaction recorded in the Fabric blockchain automatically carries the certificate information, making it larger than the transaction proposed from the vote server.

Experiment 2: Impact of data consistency process

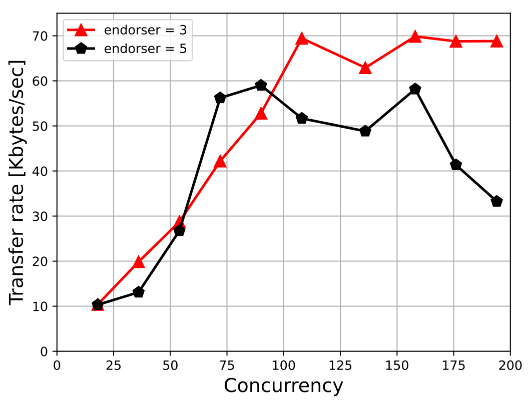
The designed on-chain and off-chain hybrid storage scheme might become the bottleneck of the system performance. We run an experiment by comparing the performance of adding this scheme in the system with the performance of removing this scheme to evaluate the impact of this design scheme. Results are depicted in Fig. 3.6(a) and Fig. 3.6(b), with the increasing number of concurrency, the throughput reaches its peak and fluctuates between 25 and 30 requests per second, and the transfer rate fluctuates between 63 and 71 KB per second.

The Fig. 3.6(c) shows the impact of the data consistency process on time taken. Overall, the time taken in Fig. 3.6(c) is not significantly different. It indicates that the data consistency process does not consume plenty of time to manipulate and verify the data consistency between the private blockchain and the public blockchain.

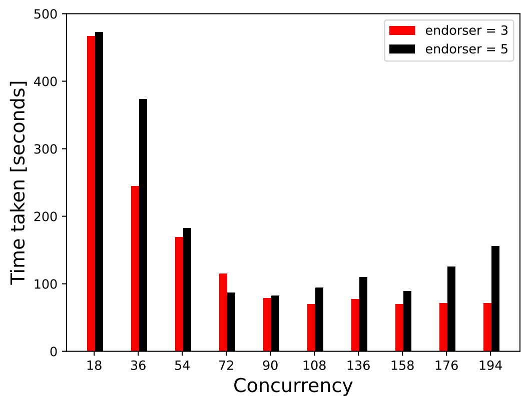
However, this design scheme can still be the bottleneck of system performance when the database read and write speed reaches the maximum value. Meanwhile, the data consistency design may take voters plenty of time to get receipts when the system fails in the data consistency process. In future work, we need to consider improving this design scheme to reduce the impact on system performance and



(a)



(b)



(c)

Figure 3.7: The impact of endorsement policy on system performance. (a) The impact of endorsement policy on throughput. (b) The impact of endorsement policy on transfer rate. (c) The impact of endorsement policy on time taken for test.

enhance the user experience.

Experiment 3: Impact of endorsement policy

When building the system, the number of endorsers and the design of endorsement policy are likely to change based on the actual government situations. To evaluate the impact of the endorsement policy, we increase the endorser number to five in each blockchain and set the different endorsement policy to satisfy the majority rule.

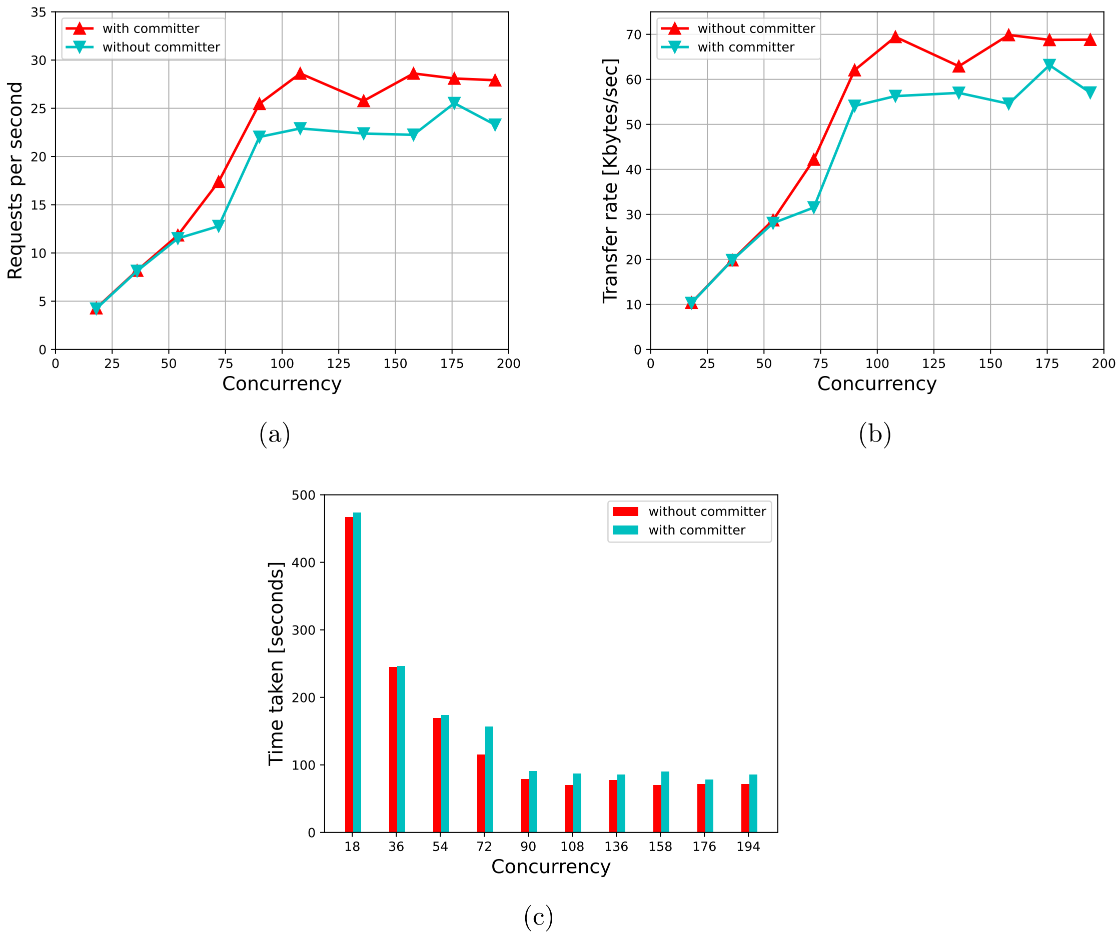


Figure 3.8: The impact of committer on system performance. (a) The impact of committer on throughput. (b) The impact of committer on transfer rate. (c) The impact of committer on time taken for test.

For reaching the consensus, three of five endorsers should simulate the same result. Then, we set the experiment to compare the system performance with the setup that the endorser number is three. Results are depicted in Fig. 3.7(a) and Fig. 3.7(b), with the increasing number of concurrency, the throughput and transfer rate reach their peak when the concurrency is equal to 90 and start to decrease.

As shown in Fig. 3.7(c), the system that each blockchain with five endorsers needs more time to finish the requests than the system with three endorsers. It indicates that the increasing number of endorsers has a significant impact on the system performance.

The increasing number of endorsers can affect the system's performance since the computing resources are not enough for the system to satisfy the endorsement policy. As the endorser number increases, more resources are needed. However, in our experiment, we set the whole peer nodes in one computer and each endorser needs to share the resource to simulate the request, which affects the performance and causes more time to finish the experiment. We believe this problem can be solved when each endorser is hosted on a single server, and their performances are not affected by each other.

Experiment 4: Impact of committer

The committer in Hyperledger Fabric is a type of peer node that has only the system chaincode to update the blockchain in its local ledger. To enhance the data security, we can set the committer in each blockchain network for retaining the copy of the ledger as a backup. Since the committer does not install any customized chaincode and its local ledger is regarded as a backup, the probability of adversary attacking committer is low. We leaves the security issues for the committer to the future work.

The experiment shows the impact on system performance when setting two committers in our system. As shown in Fig. 3.8, although the committer does not seriously degrade system performance, it still causes a non-ignorable influence on system performance.

The reason why the committer degrades the system performance is similar to experiment 3. The committer needs computing resources to update the ledger, and resources for the endorser have to be reduced since all the nodes are hosted in one computer. But, according to the experiment result, we can observe that the committer needs fewer resources than the endorser because it does not need to simulate requests. This problem can also be solved by setting the committer in a single server, in which the endorser's resource will not be occupied.

3.5.2 Security analysis

This subsection analyzes how our designed system satisfies the requirements. And the analysis is stated as follows:

Auditability

Impersonating attack [4] can disrupt the election by impersonating unqualified citizens as the legitimate voters to cast the ballots on the government server. The E-voting system should ensure the identity of each voter is authentic and legitimate. The records should keep who is involved in the transactions to audit the election result and investigate the legal issues.

In our system, the voters are instructed to complete the offline registration and online verification before joining the election. The offline registration requires the voters to submit valid personal information to the official website, including the ID number and phone number. The system produces a list of eligible voters privately for further verification. The online verification requires the voters, who have registered on the list, to provide the face ID and fingerprint to the official website. The online verifiers check the list and audit the validity of the voters' information submitted. This offline-online design leads the technologies to form an auditable verification system. Meanwhile, the records in the private blockchain contain the voters' public key ring and other personal information that can be traced back to voters when auditing elections or investigating legal issues.

Privacy

Privacy protection in E-voting is the biggest concern of voters because the leakage of privacy may threaten the personal and property safety of voters. When voters' privacy is leaked, the E-voting can be threatened by voter coercion attack [29]. The coercers can oblige the voter to vote as they wish.

The double blockchain system structure allows us to store the voting results and voter information separately. After the signed ballots have been recorded in the private blockchain, the system strips the personal private information and the public key ring to form the queryable data. Then, it is appended to a public blockchain for public supervision. The original ballots recorded in private blockchain only be used for legal purposes, and the private information will never be leaked to the public. This design keeps the voters' information confidential, and the public key ring brings anonymity for the voter when they vote.

Authentication

Suffrage or enfranchisement, the right to vote, are important but seldom mentioned in the E-voting paper until the authentication is defined. The authentication must be satisfied in our E-voting system to solve these legal issues in E-voting [62].

The system designs an identity verifier to verify the validity of voters and authorize them the right to vote. The voters must register and verify themselves by sending personal information, real-time face ID, and fingerprints to the identity verifier. After the voters pass this process, the unique linkable ring signature is generated for each valid voter, representing they have the right to vote in this election. The valid voters sign the filled ballot by their unique linkable ring signature before submitting their ballots. Moreover, the distributed, append-only and immutable properties in blockchain technology ensure that valid signed ballots cannot be changed when recorded. These technologies provide authentication in the system.

Correctness

The E-voting system must record and count the valid ballots correctly to prevent server attacks such as ballot tampering [57].

In our design, the recording and counting processes are executed in a secured, distributed blockchain-based system, preventing hackers attacks from the server-side. The peer nodes are launched in the Docker container, which supports a secure execution environment for the endorsers to execute the chaincode safely. As well as, the endorsers must comply with the specific endorsement policy when running the chaincode. The endorsement policy requires the majority of the execution results generated by different endorsers must be consistent before updating the blockchain. It prevents internal corruption from disrupting the regular operation of the system.

The system provides the query server, allowing voters to search the ballot data recorded in the public blockchain. The queried data contains the ring signature, and voters can verify the validity to prove the integrity and correctness of recorded ballots. The citizens can supervise the voting results, which have been recorded in the public blockchain, through the website supported by the system. This design prevents the government server from tampering with the ballots before being recorded in the blockchain.

The trusted executing environment, endorsement policy, and public supervision bring the recording and counting process correctness into our system.

Unreusability

The E-voting systems need to satisfy this requirement to prevent the double-voting attack [62]. The double-voting attack in the E-voting system means attackers try to clone and cast a ballot that has been cast by another valid voter previously, or the voters want to execute the voting process more than once.

In our system, this requirement is supported by the linkable ring signature. When the voters submit the signed ballots to the system, the system checks the linkability of each signature besides the validity. The linkability of the ring signature is represented by the public key image stored in the signature data structure. The public key image is generated by a hash function whose input is the real public key. According to the principle of hash collision, if two hash values are equal, the input value of the hash function is also the same. Therefore, if the signature of the submitted ballot is linkable with the signature recorded in the private blockchain, it means that the public key image of the two signed ballots is the same, thus inferring that the voter has already voted and the submitted ballot is invalid.

3.6 Conclusion

This chapter presents a DBE-voting system, which is the first BE-voting that can covers all the core requirements of a reliable BE-voting system. DBE-voting is powered by two innovative designs: double blockchain architecture, consisting of a private blockchain and a public blockchain, and an on-chain and off-chain hybrid storage mechanism that combines the off-chain database with the blockchain. Besides, we implement a prototype of our system and analyze whether our system can meet all requirements. The experimental results show that the system has better performance when the block size is 512 KB, and the data consistency method has not become the bottleneck of the experiment at the current stage. The security analysis proves that the DBE-voting can satisfy all the core requirements simultaneously.

Chapter 4

A Secure and Reliable Blockchain-based Audit Log System

4.1 Introduction

In computing, log files are responsible for recording events of an operation system or software [22], and the messages exchanged between different users can also be treated as log data. Log files have been widely used for digital forensics [3], such as tracking database tampering, versioning file system, or Internet of Things (IoT) based vehicle systems [14]. Many researchers propose various methods to generate and analyze the log file efficiently and effectively [16,34]. However, log data security problems are paid relatively little attention, especially, many of them neglect the importance of protecting the data integrity of log files.

Ensuring the data integrity for auditing the log file is crucial. For example, we can consider the scenario where a conflict happened between the server and the user. The server operates a system to supply some services, and the user may use this system through an application or something else. Log files generated during the period of user interaction with the system are stored in the system's local database, and they can be treated as digital forensics. When a conflict happened between the server and the user, they need a trusted third party, e.g., the court, to arbitrate between them. Then, the related log file, as the important evidence, can help the court judge where the responsibility lies. However, log files are controlled by the server before it sends them to the court. If the server knows in advance that it takes the responsibility for the conflict, there is a high possibility that it tampers with the log file to avoid the penalty from the court. Therefore, maintaining the data integrity for the log file is significant.

Some existing efforts are trying to maintain the data integrity for log files. These existing schemes can be separated into two categories: centralized audit log system and blockchain-based audit log system. In the centralized audit log system, log files are stored in a central database. They propose the hash-based data structure for storing the log data in a tamper-proof manner, such as trees, Rivest-Shamir-Adleman (RSA) accumulators [47], skip lists, or authenticated Directed Acyclic Graphs (DAGs). These data structures are leveraged to build certificate revocation list [11, 60], tamper-evidence graph, geometric searching [69], and authenticated responses to Extensible Markup Language (XML) queries. However, the single logger and/or auditor design in centralized log systems are vulnerable to the log injection attack [21] and collusion attack, which compromise data integrity. Meanwhile, the central database suffers from single-point failure, which prevents systems from achieving data integrity.

Blockchain technology enables secure and immutable record-keeping in a distributed system. Applied to the audit log system, blockchain can process and replicate log files correctly over a set of peers, thereby providing them with a consistent and tamper-proof view of the system. Some researchers have proposed the blockchain-based audit log systems [5, 7, 13, 18, 27, 37, 38, 43, 51, 52, 59, 66, 68]. This design increases the cost and complexity of the attack and enhances the overall defense capability of the audit log system. However, the current blockchain-based audit log systems still have two challenges.

(1) **Partial threat model and system design.** the threat models proposed by the existing systems are not general enough to defend against complicated attacks. They set a single logger and/or auditor, which can be compromised by the adversary or colluded with others. The adversary party may inject log data, return false audit results, and leak data privacy.

(2) **Data storage flaws.** They meet the problem that the blockchain is not suitable to store large-sized files [17]. With the increase of the scale of a modern system, the size of the generated log file can be very large (e.g., 22,000 events per second) and surpass the scalability of blockchain [41]. Besides, the single-point failure problem remains if the system only stores proofs in a blockchain and leaves log files in a central database.

In this paper, we propose a blockchain-based audit log system to ensure the data integrity for generated log files while addressing the remaining challenges.

First, in our threat model, we assume the logger and the auditor are untrusted machines to generate integrity proof and audit the proof. The log provider is trusted when it generates log files, but it may tamper with log data after files are stored in its central database. Our designed audit log process can detect the log tampering attack from the log provider. In our design, multiple nodes can operate the logging and auditing processes. They install the smart contract and follow the consensus algorithm to tolerate a degree of compromise and collusion attack. Compared with other approaches [7, 13, 18, 27, 37, 38, 43, 52, 59, 66, 68], our system can tolerate one-third of colluded loggers and auditors while ensuring data integrity. In addition, log providers encrypt log files before sending them to the logger to protect data privacy.

Second, to solve the data storage flaws, we extract the concept of the non-fungible token (NFT) from the Ethereum [64] and realize it in a consortium blockchain called sub-NFT. Our system can generate a sub-NFT as the integrity proof for the log file, and only these sub-NFTs are stored on the blockchain. The scalability of blockchain is capable to store small-sized sub-NFT, and log files are stored in Inter-Planetary File System (IPFS) as well as in a central database. IPFS is a distributed file system and files stored in it have a backup in multiple IPFS nodes. This design also eliminates the concern of single-point failure in a central database. Our system can resolve the single-point failure in the central database and save approximately 50% storage space for Hyperledger Fabric compared with a conventional blockchain-based audit log system [5].

- **Secure design with a general threat model.** We build a general threat model for the audit log system and propose a blockchain-based audit log system to ensure data integrity. Our system can ensure data integrity as well as detect the log tampering attack from the log provider under the circumstance of colluded loggers and auditors, providing a more secure and reliable audit log system.

- **Storage saving with an NFT-based integrity proof generation method.** We propose an NFT-based integrity proof generation method to generate and store a sub- NFT on blockchain for a log file as integrity proof. Compared with the state of the art, our method can save approximately 50% storage space for Hyperledger Fabric and resolve single-point failures in the central database.

- **System implementation and security analysis.** We build a system prototype based on Hyperledger Fabric and evaluate its performance. Security analysis proves that under the proposed threat model, our system can tolerate one-third of colluded nodes while ensuring data integrity.

The remainder of this chapter is structured as follows. Section 4.2 reviews the related work of the blockchain-based audit log file system. The system roles and notations, and threat model are defined in Section 4.3. The system model is described in Section 4.4. Section 4.5 and Section 4.6 report the evaluation results and the security analysis. Finally, this paper is concluded in Section 4.7.

4.2 Related Work

In this section, we first review some related work for the blockchain-based audit log system.

The log files are owned by the log provider and stored in its central database. In the current blockchain-based audit log systems, they set a module called logger to collect log files from the log provider, transact with the blockchain network and generate integrity proofs. The logger in [5, 7, 37, 43, 51] stores log files directly in the blockchain network and gets the transaction identification as the integrity proof. it returns the proofs back to the log provider, which stores the proofs in its local database. However, the local database storage method may cause the single-point failure problem, the scalability of blockchain is limited and it is not suitable to store large-sized files [17]. For evading the scalability problem, the logger in [13, 18, 27, 38, 52, 59, 66, 68] uses a hybrid storage method. It collects log files, generates small-sized integrity proofs of these files, such as hashes or anchor messages, and stores these data on the blockchain network. The original log files and related transaction identifications are stored in the local database of the log provider. However, the concern of single-point failure still exists in the off-chain database, and the cost of their use of public blockchains (Bitcoin and Ethereum) is expensive.

Meanwhile, the threat models in most of these systems are not general [5,18,27, 37,38,43,52,59,66]. They need to trust a single logger to collect log files, generate proofs, and store them in a central database. A single logger can be compromised by the adversary to leak privacy or launch a log injection attack, or it can also collude with another party. Moreover, relying on a single auditor to verify data integrity and achieve system feasibility is not reliable and impractical in the real world. Some works do not even have an auditor, or their auditing processes are unclear [7, 13, 37, 68].

To eliminate these aforementioned limitations, we propose a general threat model that assumes the logger and the auditor are untrusted. They can be compromised or colluded with another party to fraud others. Under this threat, we design the logging and auditing processes in our blockchain-based audit log system to ensure data integrity (details are introduced in Section 4.4). We design an NFT-based integrity proof generation method to resolve data storage flaws in existing audit log systems. Our system can generate sub-NFT as integrity proof and store this small-sized data in blockchain to resolve its limited scalability problem. Meanwhile, log files are stored in IPFS as well as in a central database, which relieves the concern of single-point failure.

4.3 System Roles and Threat Model

In this section, we introduce the system roles in our audit log system and the notations that will be used throughout the rest of this paper. Then, we construct the threat model of our system.

4.3.1 System Roles

Based on the summary of related work, we understand that the audit log system consists of multiple roles which are responsible for different functions. In the remainder of this paper, we will use the following system roles:

- *Log Provider:* The representation and abstraction of all systems that can generate different types of log files, such as web servers or cloud service providers.

- *User:* People use the client or application to interact with the *Log Provider*. The crucial log file will generate during the interaction between the *User* and the *Log Provider*. These log files may become important digital evidence when a conflict happens.

- *Logger:* The system component receives the encrypted log file, outsource it, and transacts with the blockchain to generate the integrity proof for the log file.

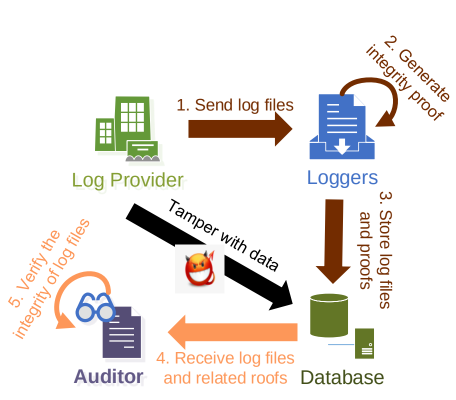
- *Court*: A third party arbitrates the conflict between the *Log Provider* and the *User*. It will receive the log file and corresponding proof from the *Log Provider*. The log file is treated as digital evidence, and the proof is used to verify the log data integrity.

- *Auditor:* The system component to verify the data integrity of the log file sent by the *Court*.

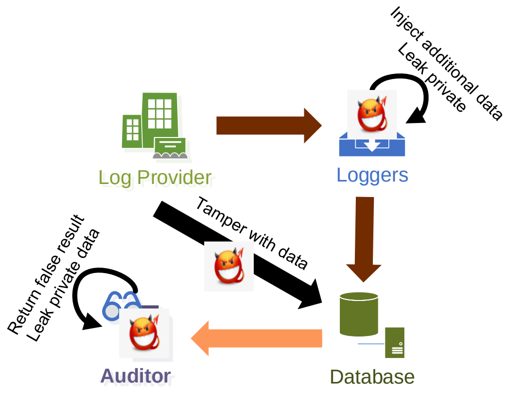
For these five system roles. we mainly discuss and analyze the *Log Provider*, *Logger* and *Auditor*. They compose the entire audit log system and realize the fundamental processes of file generation, proof generation, and proof verification. The *User* and *Court* are also important and essential for the system, but they are not the focus of this paper. We set up these two roles to have a comprehensive security analysis for the *Log Provider*, *Logger*, and *Auditor*, and to be close to realistic scenarios.

4.3.2 Threat Model

Different from the conventional audit log systems which need to trust the single logger and/or auditor (as shown in Fig. 4.1a), in our system we consider that the


1. Send log files
2. Generate integrity proof
Log Provider
Loggers
3. Store log files and proofs
Tamper with data
5. Verify the integrity of log files
4. Receive log files and related roofs
Auditor
Database

(a)



(b)

Figure 4.1: The comparison of the threat model of conventional audit log systems with that of our system. (a) The threat model in conventional audit log systems. (b) the threat model in our system.

Log Provider, *Logger*, and *Auditor* are not the trusted machine (as shown in Fig. 4.1b). The states and malicious behaviors of our system roles are described as follows.

- The *Log Provider* can be compromised. It may tamper with the log files, which are stored in its local database, to avoid the penalty. It can also operate in collusion with the *Logger* or *Auditor* to defraud the other members. Finally, due to the unpredictable malfunction that happened in the hardware, the single-point failure, and the data loss concerns in the *Log Provider* central database are un-negligible.

- The *Logger* and *Auditor* can be compromised. They may leak the private data recorded in the file, and the *Logger* can inject additional information into the log file that it is processing. Meanwhile, the *Log Provider* may collude with the *Auditor* to defraud the other member. For example, the *Log Provider* can collude with the *Auditor* to return the false audit result of the log file that the *Court* received from the *Log Provider*. Finally, the single-point failure needs to be concerned in the *Logger* and *Auditor*.

- The *User* and *Court* are fully trusted in our system.

It is important to note that the *Log Provider* is trusted when it generates log files. According to the [55], we know that there is no security measure that can protect log files generated after the system has been compromised. So, we assume the *Log Provider* is trusted at the log file generation stage. We also regard the data recorded in the Hyperledger Fabric is immutable, and the blockchain is hard to fork.

4.3.3 System Notations

In the remainder of this paper, we will use the following notations:

- log_i represents the ith log file sent from the *Log Provider*.
- $E_{K_s}(log_i)$ is the secret key encryption, under the *Log Provider* secret key, of log_i, using Advanced Encrypting Standard (AES) [19] algorithm with a key length such as 128 bits and scheme such as Cipher Block Chaining (CBC).
- CID_i is the content identifier of log_i, indicating the log_i has stored in the IPFS.
- $TxID_i$ is the transaction identifier of CID_i, showing the *Logger* makes the transaction with the blockchain.
- $E_{PK_c}(K_s)$ is the public-key encryption, under the *Court* public key, of K_s using an algorithm such as RSA [53].
- $True_i$ or $False_i$ is the result of integrity verification process on log_i. The $True_i$ means the data integrity of log_i is maintained, and the $False_i$ means not.

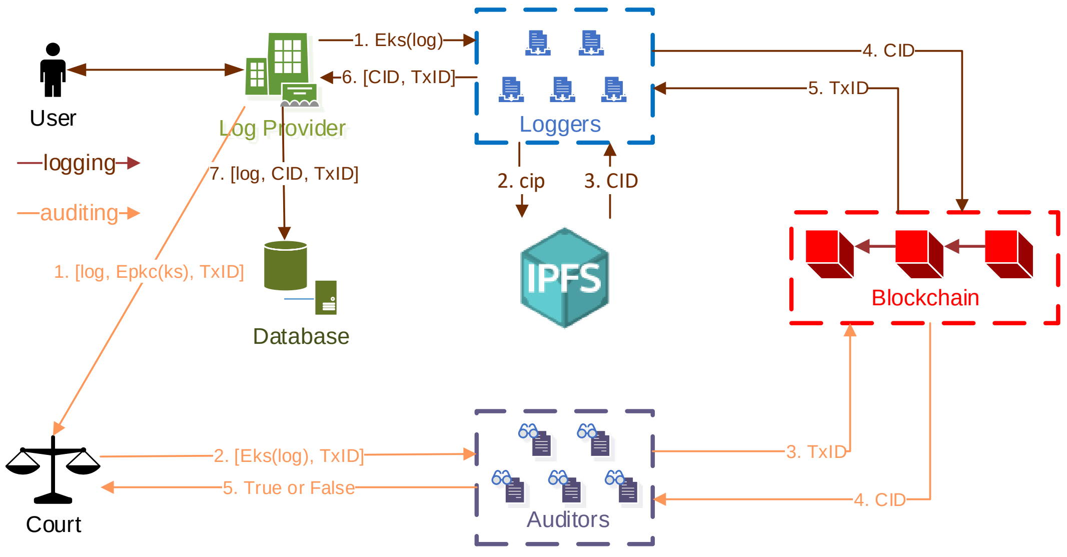


Figure 4.2: The proposed blockchain-based audit log system model.

4.4 System Method Description

In this section, we propose the system model of our blockchain-based audit log system and describe our method in detail.

As shown in Fig. 4.2, our system has two different processes: the logging process and the auditing process. In the logging process, some important log files are generated during the interaction between the *User* and the *Log Provider*. Then, the *Log Provider* generates the data integrity proof for each log file by requesting the *Logger*. During the logging process, the *Logger* outsources the encrypted log files to IPFS to solve the single-point failure problem and generate CID as integrity proof. Finally, the original log file and related proof information are stored in the *Log Provider* local database for other proposes like data analysis and system check. When the conflict happened between the *User* and the *Log Provider*, the auditing process is launched. During this process, the *Court*, as the trusted third party, arbitrates this conflict, and it receives the log file from the *Log Provider* as the important digital evidence. Before the arbitration, the *Court* launches the auditing

Algorithm 2 Sub-NFT generation chaincode in the *Logger*

```
Input: Encrypted log file $cip_i$, chaincode stub interface $stub$.
Output: Content identifier $CID_i$, transaction identifier $TxID_i$.
1: Connect IPFS peer node: $sh = NewShall($"IPFS API"$)$
2: Add file to IPFS and get CID: $CID_i = sh.AddFile(cip_i)$
3: Get current $TxID_i$ from Hyperledger Fabric:
4: $TxID_i \leftarrow stub.GetTxID()$
5: Generate $key$ of sub_NFT: $key \leftarrow$ "CID" $+TxID_i$
6: Generate sub_NFT by recording $key$ and $CID_i$ as a pair in Hyperledger Fabric:
   $stub.PutState(key, CID_i)$
7: return $TxID_i, CID_i$
```

process by requesting the *Auditor* to verify the data integrity of the receiving log file.

In the following subsections, we will describe the logging process and auditing process in detail.

4.4.1 Logging Process

- **Setup**: The *Log Provider* generates the secret key K_s randomly, which key length can be 128 bits or 256 bits.

- **Encrypt**: Before interacting with the *Logger*, the *Log Provider* encrypts the log file, e.g. log_i, by the secret key K_s and generates the ciphertext $cip_i \leftarrow E_{K_s}(log_i)$.

- **Generate**: After the encryption, the *Log Provider* sends the cip_i to the *Logger* to generate the data integrity proof for the encrypted log file cip_i. The *Logger* executes the chaincode to generate the sub-NFT as the integrity proof, outsource the cip_i in IPFS and return the result to the *Log Provider*. In this

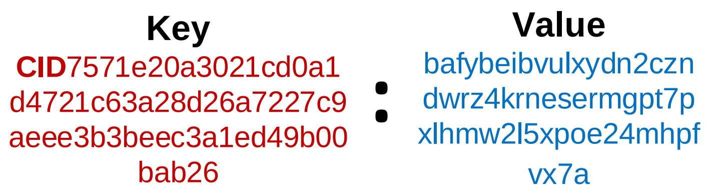


Figure 4.3: An example of a key-value pair for a sub-NFT recorded in Hyperledger Fabric.

example, the result is $(TxID_i, CID_i)$.

- **Store**: If the *Logger* executes the chaincode successfully and the *Log Provider* receives the result, the *Log Provider* combines the log file and the related proof information as a triple and stores it in its local database. In this example, the triple is $(log_i, TxID_i, CID_i)$

The Algorithm 2 shows the detail of the chaincode in the logging process. The peer node in the *Logger* installs this chaincode and launches the transaction. In this chaincode, the peer node first builds a connection with the IPFS server by calling the $NewShall()$ function and uploads the cip_i to IPFS. If the peer node successfully adds cip_i to IPFS, we can get the corresponding CID_i. Then, the peer node can get the current transaction identifier $TxID_i$ and combines it with the string "CID" to form a specific key. In Hyperledger Fabric, the transaction identification is genereated when invoking the chaincode, so it can be obtained by calling the $GetTxID()$ function. Finally, the peer node generates the sub-NFT by recording the key-value pair in the blockchain.

Since we assume the logger is untrusted in our threat model, we utilize blockchain technology and design NFT-based integrity proof generation method in our system to defend against adversary attacks in the logger. As shown in Fig. 4.2, we set multiple peer nodes in the *Logger*, and these nodes all install the same chaincode

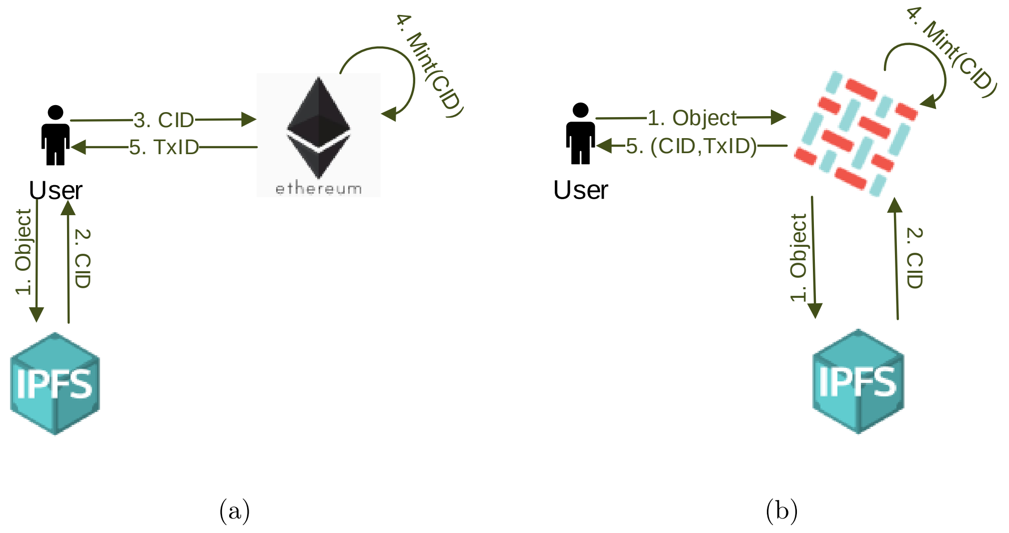


Figure 4.4: The conclusion of NFT and sub-NFT generation processes. (a) The NFT generation process in Ethereum. (b) The sub-NFT generation process in our Hyperledger Fabric based system.

Table 4.1: Comparison between NFT and sub-NFT.

	Existence	Ownership	Trade	Cost
NFT in Ethereum	√	√	√	High
Sub-NFT in our system	√	×	×	Low

to generate sub-NFT as the data integrity proof. All the peer nodes in the *Logger* follow the endorsement policy to reach the consensus and return the valid transaction. Section 4.6 proves our system can defend against attacks from compromised or colluded logger nodes.

Fig. 4.4 shows a comparison of the NFT generation process in Ethereum and the sub-NFT generation process in our system and Table 4.1 summarizes the main different properties between NFTs and sub-NFTs. We can see both NFT and sub-NFT are minted by launching smart contracts in different blockchains; blockchain network stores CIDs and IPFS stores objects, but the process of outsourcing objects

Algorithm 3 Audit chaincode in the *Auditor*

Input: Encrypted log file cip'_i, transaction identifier $TxID'_i$.

Output: Audit result $True_i$ or $False_i$.

```
 1: Initializes key-value pair data structure to store transactions: $kvpair \leftarrow map[string]string$
 2: Get the specific Hyperledger Fabric block by querying the $TxID'_i$: $block \leftarrow QueryBlockByTxID(TxID'_i)$
 3: $key \leftarrow$ "CID" $+TxID'_i$
 4: Get all transactions in the block: $tx \leftarrow GetTransaction(block.Data)$
 5: for $k, v \in tx$ do
 6:     $kvpair[k] = v$
 7: end for
 8: Get the $CID_i$ that is recorded in Hyperledger Fabric: $CID_i \leftarrow kvpair[key]$
 9: Generate $CID'_i$ from $cip'_i$: $CID'_i \leftarrow cid.Perfix.Sum(cip'_i)$
10: Audit the input log file data integrity:
11: $result \leftarrow CID_i.Equal(CID'_i)$
12: if $result = true$ then
13:     return $True_i$
14: else
15:     return $False_i$
16: end if
```

is executed by different roles. A similar generation method ensures the existence of sub-NFTs to prove the integrity of corresponding objects. NFTs generated in Ethereum are signed by their owner to ensure ownership, however, it is unnecessary for sub-NFTs since all log files are generated by the same *Log Provider*. Meanwhile, sub-NFTs cannot trade since they do not have transaction value. Finally, NFTs generated in Ethereum cost expensive gas fees, and sub-NFTs generated in our system do not require money, but computational resources.

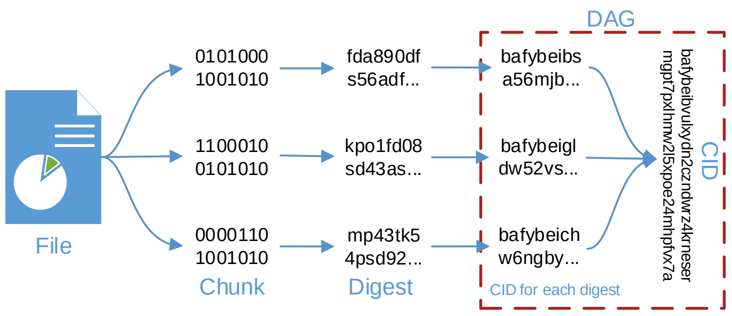


Figure 4.5: The generation process of CID.

4.4.2 Auditing Process

- **Setup**: The *Court* chooses a key pair (PK_C, SK_C), as the private key and public key.

- **Send**: The *Log Provider* sends the log file log_i' as the digital evidence, to the *Court* and sends the related $TxID_i'$ as the proof information. In addition, since the *Logger* generates the sub-NFT for the encrypted log_i', the *Log Provider* sends its secret key K_s' to the *Court*, which the K_s' is encrypted by the *Court* public key PK_C.

- **Encrypt**: Before auditing the data integrity proof, the *Court* encrypts the log_i' by the K_s' and generates the ciphertext $cip_i' \leftarrow E_{K_s'}(log_i')$.

- **Audit**: After the encryption, the *Log Provider* sends the cip_i' and $TxID_i'$ to the *Auditor* to verify the data integrity of log_i'. The *Auditor* launches the chaincode to generate the CID_i' locally, query the corresponding CID_i recorded in the blockchain through $TxID_i'$, compare the CID and return the result to the *Court*. The result is $True_i$ or $False_i$.

The Algorithm 3 shows the detail of the auditing process. The peer node in the *Auditor* installs the chaincode, accesses the record in the blockchain, and audits the file integrity. In this chaincode, the $Auditor$ first calls the $QueryBlockByTxID()$ to locate the specific block which stores the transaction with the $TxID_i'$ as the transaction identifier. The $block.Data$ data structure contains all the key-value pairs in that block and the $Auditor$ extracts all the pairs by calling the $GetTransaction()$ function. Then, the $Auditor$ finds the specific CID_i whose key is the combination of string "CID" and $TxID_i'$. Finally, the $Auditor$ can generate the content identifier CID_i' of the cip_i' locally and compare the hash value of these two identifiers. If they are equal, the $Equal()$ function returns $true$ value and the $Auditor$ returns the $True_i$ to indicate the data integrity of log_i' is maintained. If they are not, the $Auditor$ returns the $False_i$.

Fig. 4.5 illustrates how the function in Algorithm 3 generates CID. First, the function divides the file into multiple chunks and each size is 256 KB. Then, it digests each chunk into a unique hash code by a specific hash function. Next, it further edits and processes each digest, and generates the unique CID for each chunk. Finally, it aggregates all the chunk CID by forming a DAG data structure and generates only CID which is the unique identifier for the input file. This file CID can help our system locate the file content in IPFS and audit the data integrity.

In this section, we describe how the $Auditor$ verifies log file integrity proofs and checks whether they are tampered with by the *Log Provider* or the *Logger*. And, similar to the *Logger*, the $Auditor$ also has multiple peer nodes which install the same chaincode and follow the designed endorsement policy to reach the consensus. Section 4.6 proves our system can defend against attacks from compromised or colluded auditor nodes.

4.5 Performance Evaluation

We build a prototype of our system. In this section, we evaluate the performance of some system processes.

The performances of different system processes are affected by different factors. For example, our system is implemented by Hyperledger Fabric. According to [6], we know that the Hyperledger Fabric is a complex distributed system, and its performance is related to many parameters, including the system node topology, network bandwidth, query efficiency, and hardware. IPFS is utilized in our system to back up log files and generate integrity proof. The time costs of uploading files and generating CID can affect the system's performance. Meanwhile, the AES encryption algorithm takes time and computation resources to encrypt and decrypt different sizes of log files, which also affects system performance.

Since the *Log Provider*, the *Logger*, and the *Auditor* are the main roles in our system and each is responsible for a specific system process, our experiments are separated into three parts: encryption and decryption performance, upload and integrity generation performance and integrity proof performance. In each part, we set experiments to evaluate the performance of the system process and analyze why we get these experimental data.

The results demonstrate that our system can save approximately 50% storage space for Hyperledger Fabric compared with the work in [5]. The AES algorithm is efficient to process large-size log files in our system. When the log file size exceeds 28.95 MB, the performance of the logging and auditing process needs further improvement in future work.

Table 4.2: Brief introduction of log files.

File Name	Number of Event	Size of Original File	Size of Encrypted File
Log_100	100	217 KB	289 KB
Log_2000	2000	4.3 MB	5.8 MB
Log_4000	4000	8.7 MB	11.6 MB
Log_6000	6000	13 MB	17.4 MB
Log_8000	8000	17.4 MB	23.2 MB
Log_10000	10000	21.7 MB	28.9 MB

4.5.1 Setup

In our experiments: (1) nodes run on Fabric v1.4.2-preview instrumented for performance evaluation through local logging, (2) nodes are hosted in docker containers as dedicated VMs, (3) all nodes are running on a single macOS with Inter Core i9 2.4 GHz 8-vCPU, 32 GB of RAM and SSDs as local disks, (4) a single-channel ordering service runs a test solo with one Fabric orderer, (5) there are six peers in total, all belonging to different organizations, and the *Logger* and the *Auditor* have three peers as endorsers for each, (6) the endorsement policy is designed as *OR(AND(LoggerOne, LoggerTwo), AND(LoggerOne, LoggerThree), AND(LoggerTwo, LoggerThree))*, which satisfies the majority rule, and (7) the INFURA [1] provides IPFS service to backup the encrypted log file and generate CID.

4.5.2 Methodology

In Section 4.3.1, we suppose the *Log Provider* is the representation and abstraction of all systems that can generate different types of log files. To facilitate the implementation of experiments, we choose L2TAP privacy logs as the log file used in experiments. L2TAP provides a set of classes and properties that can be used to represent and publish log events, such as log initialization events, participant registration events, and privacy events [54]. According to [41], we know the modern system can generate a large log file in one second, which may contain over ten thousand events. In our experiments, we assume the *Logger* and the *Auditor* need to handle log files that contain multiple events. The size for each L2TAP log event is approximately 2 KB, and the AES CBC encrypt algorithm may also enlarge the size of log files by padding them. The Table 4.2 shows some details of L2TAP log files, and we can observe the size and the contained log event number in different log files that we used in experiments. In addition, the BatchTimeout, as a system parameter in Hyperledger Fabric, can be treated as the response time when the *Log Provider* or the *Court* gets the result, and we set this parameter as 2 seconds as the default value.

4.5.3 Encryption and Decryption Performance

Before the *Log Provider* sends the log file to the *Logger*, it needs to encrypt the log file for privacy protection. According to [42], we know the AES-128 is secure enough for any type of commercial application, and the security level of AES-256 is beyond anything required by the ordinary application. Meanwhile, there are two patterns for the *Log Provider* generating log file: separated pattern and combined pattern. For the separated pattern, one log file contains one event; for the combined pattern, one log file contains multiple events. The encryption algorithm with different key lengths and different log file generation patterns may affect system performance. We run experiments to compare the performance of encryption and decryption by AES-128 and AES-256 in different log file generation patterns.

Fig. 4.6 shows the time taken for the file encryption and decryption under the separated pattern with different key lengths. The encryption and decryption time increases as the number of log files increases. The reason why the encryption time

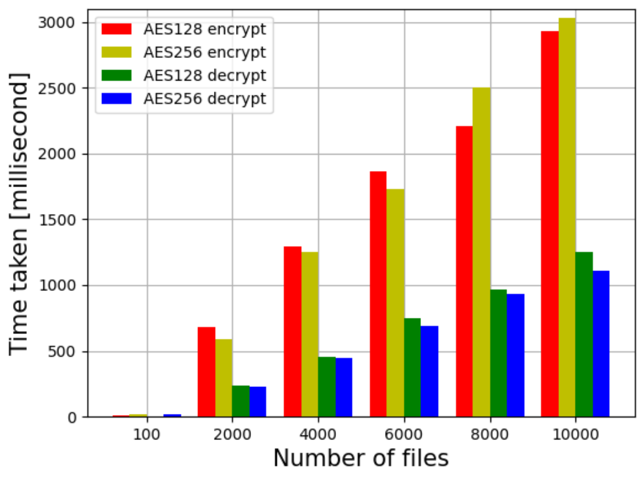


Figure 4.6: The time taken for separated files encrypt and decrypt.

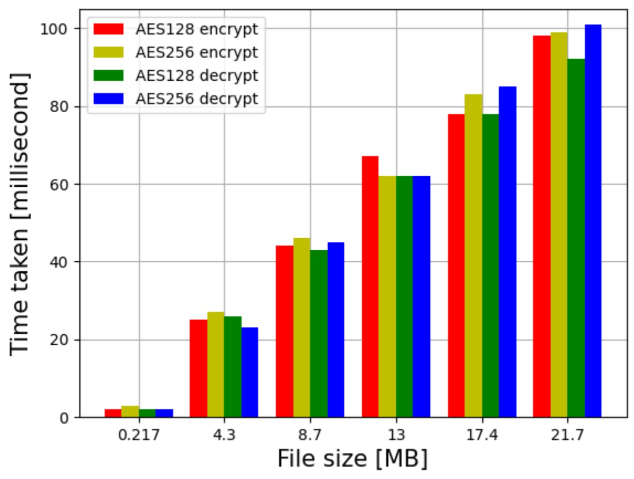


Figure 4.7: The time taken for combined files encrypt and decrypt.

is larger than the decryption time is log files are firstly read from the hard disk. In this experiment, we create and store log files on the hard disk beforehand. The *Log Provider* needs to access multiple times to encrypt each log file and generate and store the encrypted file in the hard disk for follow-up experiments. Since the decryption performance evaluation is launched after the encryption performance evaluation, the *Log Provider* may read encrypted files from the cache rather than the hard disk and this is time-saving.

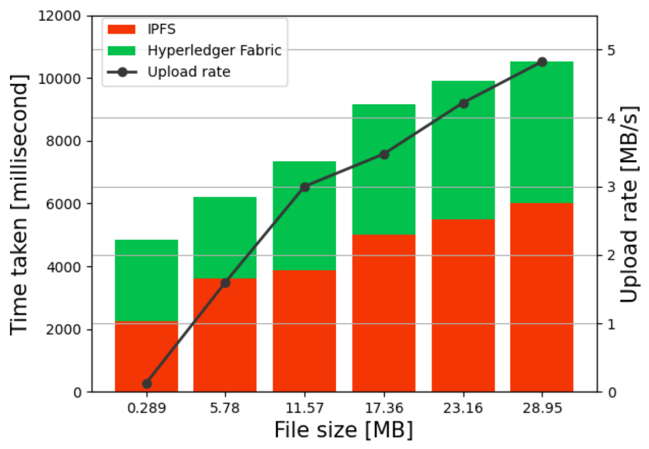


Figure 4.8: The time cost and upload rate of log files encrypted by AES-128.

Fig. 4.7 shows the time taken for the file encryption and decryption under the combined pattern with different key lengths. Similarly, we generate and store log files that contain multiple events on the hard disk; the encryption and decryption time increases as the size of the log file increases. Compared with Fig. 4.6, the time taken is reduced significantly in this experiment. The reason is the *Log Provider* only needs access hard disk once in each test.

Through this part of the experiment, we think the normal large-scale systems can utilize the AES algorithm to encrypt log files before sending them to the *Logger* since this algorithm has little effect on system performance. From Fig 4.7, we can observe that the time taken of encrypting and decrypting a log file that contains 10,000 events are no more than 105 ms. We believe this resource cost is acceptable for normal large-scale systems, and this cost can be reduced if the *Log Provider* reads and encrypts the log file as soon as it is generated, rather than waiting for it to be stored on a hard disk.

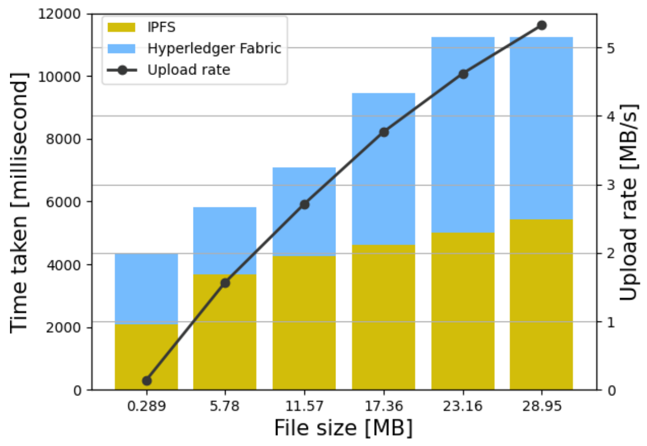


Figure 4.9: The time cost and upload rate of log files encrypted by AES-256.

4.5.4 Upload and Integrity Proof Generation Performance

After the *Log Provider* encrypted log files, the *Log Provider* sent them to the *Logger* for outsourcing log files and generating integrity proof. Fig. 4.8 shows the performance of executing combined log files encrypted by AES-128 and the log files in Fig. 4.9 are encrypted by AES-256. The size of log files encrypted by different key lengths are the same, and these two experiments have the same logging procedure, so we use Fig. 4.8 as an example to analyze this part of the experiment.

In Fig. 4.8, we can observe the time taken for the logging process increases as the size of the encryption log file increases. The entire system process is executed by chaincode in Hyperledger Fabric, but it can be roughly divided into two parts: IPFS and Hyperledger Fabric. In detail, we can observe the time taken occupied by IPFS and the upload rate increase as the size of the encrypted log file increases. We think the time taken in Hyperledger Fabric is mainly cost by peer communication and execute-order-validate procedure. Since we set the BatchTimeout is 2 seconds and the size of the encrypted log file is increasing, all the time taken in Hyperledger Fabric is larger than 2 seconds and keeps increasing.

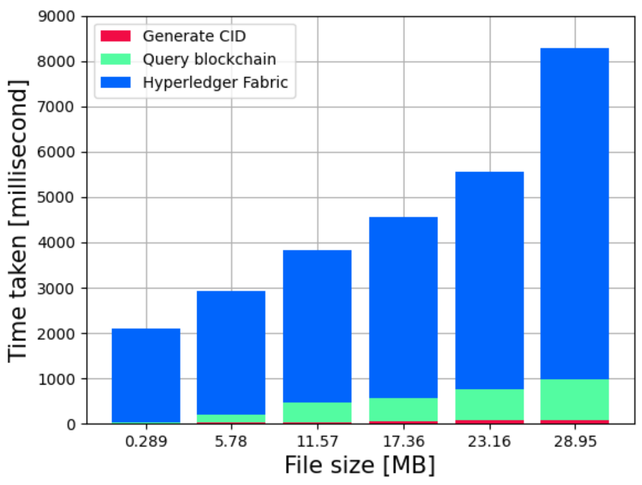


Figure 4.10: The time of integrity proof for log files encrypted by AES-128.

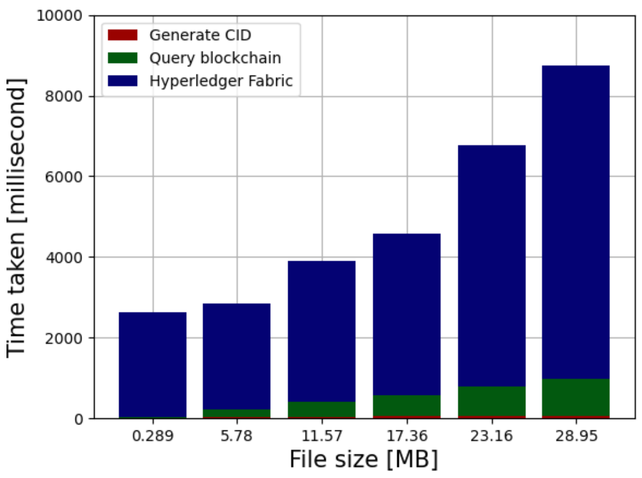


Figure 4.11: The time of integrity proof for log files encrypted by AES-256.

4.5.5 Integrity Proof Performance

This part of the experiment evaluates the performance of the auditing process when the *Court* receives a log file and wants to verify its data integrity. Section 4.4.2 introduces the auditing process in the *Auditor*, and Fig. 4.10 and Fig. 4.11 show the performance of auditing different encrypted log files. As aforementioned, the size of log files encrypted by different key lengths and the auditing procedure are the

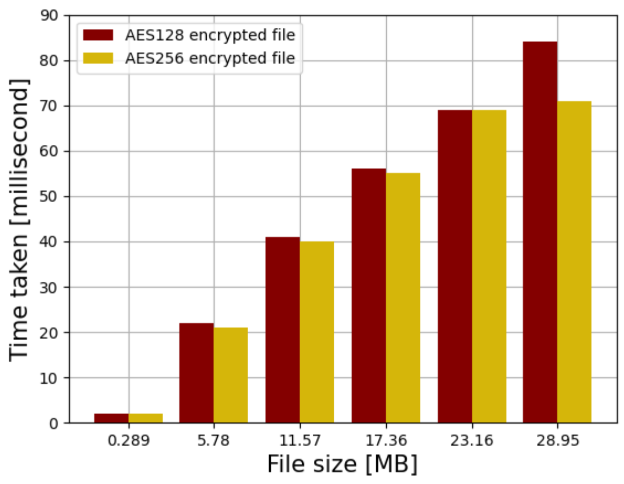


Figure 4.12: The time of generating cid for different log files.

same, so we choose Fig. 4.10 as an example to analyze this part of the experiment.

As shown in Fig. 4.10, the time taken for auditing process increases as the size of the encrypted log file increases. Similar to the experiment in Section 4.5.4, we roughly separate process into three parts: generate CID, query blockchain, and Hyperledger Fabric. Specifically, Fig. 4.12 clearly shows the time taken to generate CID for different encrypted log files. It is obvious that the peer node needs to take more time to generate CID for the encrypted log file if it is larger than the previous file. We consider the performance of generating CID locally acceptable since it takes no more than 85 ms to generate the CID for a log file containing 10,000 events.

The query time shows in Fig. 4.10 is increased as the file size increased. The query process in Hyperledger Fabric is different from other blockchains by setting up a Leveldb to store the addresses of blocks and their corresponding transactions. This design avoids traversing the entire blockchain and saves time. In this experiment, we evaluate each performance in the order of file size. The addresses of the block and the transaction that record a larger file are stored deeper in the database and it takes more time to get them by the sequential query.

The time taken in the Hyperledger Fabric part includes the time of other Hy-

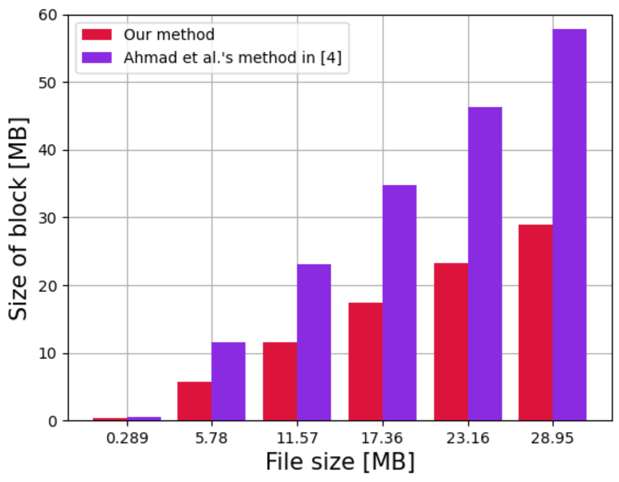


Figure 4.13: The size of block when it stores log files or related integrity proofs (the maximum block size is fixed and large enough).

perledger Fabric processes, which are the same as the processes in Section 4.5.4. Generally, the time taken in this part increases as the size of the log file increases. From Fig. 4.10 and Fig. 4.11, we can conclude the longest time to audit the largest file in experiments is less than 9 seconds.

4.5.6 Blockchain Scalability

The scalability of blockchains is limited and they are not suitable to store large-sized files. With the increase of the scale of the *Log Provider*, the size of the generated log file may surpass the scalability of blockchains. In our system, we design an NFT-based integrity proof generation method to resolve this problem. Our system only stores the sub-NFT, not the original encrypted log file, on blockchain as an integrity proof, and Fig. 4.13 shows that our method, compared with other [5], can significantly reduce the storage pressure caused by the limited scalability of blockchain.

Ahmad et al. [5] propose an audit log system by utilizing blockchain to ensure log file integrity. Both our system and their system are implemented by Hyperledger Fabric, but the main difference is they store the original log file on a blockchain. In this experiment, we evaluate two methods by comparing the size of two blocks. The block in the first test stores related integrity proofs, and the block in the second test stores log files. For comparing easily, each block only contains one transaction that records the log file or integrity proof; the maximum block size, a Hyperledger Fabric parameter, is the same and large enough.

As shown in Fig. 4.13, our method can save nearly half of the blockchain storage space and reduce its burden significantly. The reason the block size increases with the file size is that according to the design of Hyperledger Fabric, each valid transaction contains multiple pieces of information, such as key-value pairs, simulation I/O, signatures, etc. The simulation input is an original encrypted log file in our and Ahmad et al.'s Hyperledger Fabric transaction. It also explains why the block size in our system is close to the size of the input file, even though the size of the recorded sub-NFT is small, and why the block size in Ahmad et al.'s system is more than twice the size of the input file.

4.6 Security Analysis

In this section, we analyze the system security to demonstrate that our system can ensure log file data integrity under the proposed threat model.

4.6.1 Data Privacy

The information recorded in the log file can be private and sensitive for the *User* and the *Log Provider*. In the threat model, we define the *Logger* and the *Auditor* can be compromised, and they may leak the private data recorded in the log file. To protect the data privacy, we require the *Log Provider* and the *Court* to encrypt the

Algorithm 4 PKCS7Padding method for the AES

Input: Plaintext *data*, block size *size*.

Output: Plaintext with padding content $data'$.

```
1: function PADDING($data$)
2:     $padtext \leftarrow []byte$
3:     $length \leftarrow len(data)$
4:     $padding \leftarrow size - length\%size$
5:     for $i = 0 \rightarrow padding - 1$ do
6:         $padtext[i] = padding$
7:     end for
8:     $data' \leftarrow append(data, padtext)$
9:     $return\ data'$
10: end function
```

log file before sending it to the *Logger* or the *Auditor*. For the encryption algorithm, we choose the AES [19], a symmetric encryption algorithm, to encrypt the log file with different key lengths (128, 192, and 256 bits). According to [42], we know that, in theory, AES-128 can keep any type of commercial application in secure, and the security level higher than AES-128 and AES-192 exceeds any requirements for common applications. When the *Logger* or the *Auditor* receives the ciphertext that is encrypted by the AES algorithm, it is impossible for them to deduce the plaintext from the ciphertext or decrypt the ciphertext without the secret key. Therefore, the data privacy of the log file can be protected.

The AES encryption algorithm implemented in our system has some specialized settings that are used in conjunction with other security goals.

- Cipher Block Chaining (CBC): This is an encryption pattern in AES. In this pattern, the plaintext is first divided into several segments, and then each segment is XORed with the initial block or the ciphertext of the previous segment, and then encrypted with the secret key. It conceals plaintext structure information and, therefore, it is not easy to be attacked [24]

- PKCS7Padding: This is a padding method to make the plaintext 128 bits long or multiple of it since the AES algorithm encrypts the plaintext by splitting it into blocks and each block length is 128 bits. The Algorithm 4 shows that the padding content is specific and unchanged for the given plaintext. This property ensures the ciphertext is unchanged for the given plaintext that is encrypted by the same secret key.

Definition 1. *(E_K, AES encryption with PKCS7Padding) For the AES encryption function E_{K_S} which follows the PKCS7Padding pattern satisfies the property that:*

The plaintext t_i and t'_i, the corresponding ciphertext $c_i = E_{K_S}(t_i)$ and $c'_i = E_{K_S}(t_i)$. If $t_i = t'_i$ and use the same secret key K_S, then we have $c_i = c'_i$.

4.6.2 Data Integrity

According to the threat model, the log file can be tampered with by the *Log Provider* in its local database or by the *Logger* in the logging process. Our system applies blockchain technology and sub-NFT as integrity proof to detect or prevent tampering attacks. Before the analysis, some definitions need to be declared.

Definition 2. *(CID, content identifier) It is a hash value based on file contents encryption and the unique representation of the file contents. The function $sum()$ is defined to calculate the CID of the input file. If the file F_1 and F_2 have exactly same contents, then $sum(F_1) = CID_1 = sum(F_2) = CID_2$.*

Definition 3. *(TxID, transaction identifier) It is a hash value automatically generated by the Hyperledger Fabric for each transaction. TxID is unique for each transaction, even if its information and value may duplicate those previously recorded.*

$TxID_i$ represents the transaction identifier for the ith transaction and it has a property that if $i \neq j$, then $TxID_i \neq TxID_j$.

Log Tampering Detection

When the conflict happened between the *Log Provider* and the *User*, the *Log Provider* may tamper the log file in its local database to evade punishment from the *Court*. To detect this log tampering attack, the *Court* can require the *Log Provider* to send the TxID besides the log file. Then, the *Court* can verify the data integrity of the log file by sending the log file and TxID to the *Auditor*.

Theorem 1. *We assume the CID' is calculated from the log file, and CID is searched by the $TxID$. If $CID' = CID$, then the log file that the Court received is not tampered with by the Log Provider.*

Proof. We assume the logging process and the auditing process are operated correctly (the proof is in Section 4.6.2 and Section 4.6.2). The relationship between TxID and CID can refer to the logging process in Section 4.4.1. The *Auditor* can search the CID recorded in the blockchain through the TxID sent by the *Court*. If the queried CID is not null, according to the Definition 3 and the threat model, we believe the *Log Provider* provides integrity proof for a particular log file. Then, the *Auditor* calculates the CID' for the log file sent by the *Court* and compares the CID and CID'. If $CID = CID'$, according to the Definition 2 we affirm the log file that the *Court* received is not tampered with by the *Log Provider*. □

Log Tapmering Preventation

According to the threat model, the *Logger* in our system is not the trusted machine, and it can be compromised to tamper with the log file even if the *Log Provider* is trusted at this stage. To prevent the *Logger* from tampering with the log file when the *Logger* processes it, we utilize the technologies in Hyperledger Fabric.

Specifically, the node that executes the logging process is called an endorser in the Hyperledger Fabric. It installs the chaincode, which contains the logic and code of the logging process, and executes the chaincode in a secure execution environment supported by the Docker container [12]. Moreover, in our system, we set multiple endorsers in the *Logger*, and they follow the designed endorsement policy to reach a consensus when they execute the same transaction. The endorsement policy requires the majority of the endorsers to execute the same result before updating the blockchain.

We assume all the endorsers in the *Logger* install the correct chaincode successfully at the setup stage. Since the endorser is launched in a trusted executing environment, the only way for the adversary to compromise the *Logger* is by rewriting the chaincode and reinstalling it to a majority of endorsers. However, it is impossible for the adversary to shut down and reset a majority of endorsers without getting attention from the system. Therefore, the *Logger* tampering with logs is prevented.

Operation Correctness

In the threat model, we assume the *Auditor* can collude with the *Log Provider*. The *Auditor* can return a false audit result to the *Court* even if it detects the log data integrity is destructed. Therefore, we need to achieve operation correctness to defend against collusion attack when verifying data integrity.

The security goal of the operation correctness is tolerating a degree of collusion between the *Log Provider* and the *Auditor*. Specifically, the *Log Provider* can collude with the *Auditor* to return the false audit value to the *Court* and the *Log Provider* can avoid the penalty. To tolerate a degree of this collusion, we apply blockchain

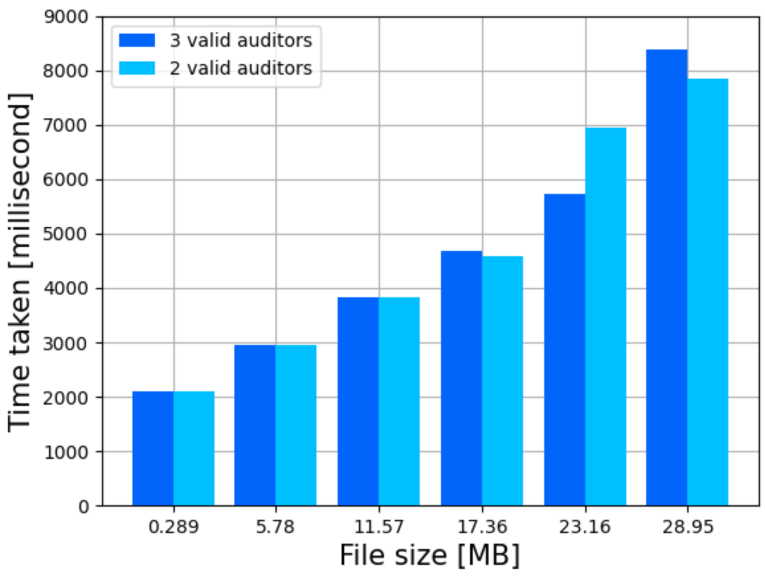


Figure 4.14: Integrity proof time for log files encrypted with AES-128 under one-third collusion attack. The deep blue bars represent the performance under a normal situation and the light blue bars represent the performance under attack.

technologies in Hyperledger Fabric.

The measure for the *Auditor* is similar to the *Logger* in Section 4.6.2. The *Auditor* node installs the designed chaincode, becomes the endorser, and executes the program in a trusted executing environment. Moreover, we set multiple endorsers in the *Auditor*, and they follow the specific endorsement policy to reach a consensus before returning the audit result.

The endorsement policy in Hyperledger Fabric is a boolean expression to guide endorsers on how to determine whether the transaction is approved or not. In our system, we require the audit result to be valid when a majority of endorsers in the *Auditor* approve this result correctly. For example, there are three endorsers in the *Auditor* and the endorsement policy is *OR(AND(AuditorOne, AuditorTwo), AND(AuditorOne, AuditorThree), AND(AuditorTwo, AuditorThree))*. In this example, the audit result is valid if two of three endorsers approve the result correctly which means the *Auditor* can tolerate the collusion attack when only one endorser has colluded. Therefore, if we want the *Auditor* to be more tolerant of the collusion attack, we can set more endorsers and scale up the endorsement policy. In addition, the *Logger* can also tolerate a degree of collusion with the *Log Provider* since the *Logger* has the same design pattern as the *Auditor*.

Fig. 4.14 shows the result that our system tolerates one-third of collusion attack when *Auditor* and *Logger* have three peer nodes in each. We choose the auditing process, where the *Auditor* verifies the integrity of log files encrypted by AES-128, as a representation. In this experiment, the *Auditor* follows the setup in Section 4.5 and we assume one scenario that the *AuditorThree* node has been colluded and cannot work properly during the auditing process. The performance with three valid auditors represents the performance of the auditing process under a normal situation and the performance with two valid auditors represents the performance under attack. Comparing the normal situation with this scenario, Fig. 4.14 shows the performance of the *Auditor* attacked by one-third of colluded nodes is not significantly different from the performance where all nodes are normal.

Data Recovery

The goal of data recovery is to reduce concerns about single-point failure when our system attempts to ensure and prove data integrity.

The *Log Provider* can store the generated log file in its local database for data analysis or digital forensics. However, the log data in its central database may be lost due to various malfunctions. For recovering those lost data, we design the *Logger* to outsource the encrypted log file in the distributed file system, e.g., IPFS and require the *Log Provider* to store the triple instead the single lof file in its local database. The example of the triple is $\{log_i, CID_i, TxID_i\}$.

Theorem 2. *We assume the triple* $\{log_i, CID_i, TxID_i\}$ *has lost some data. The* *triple can be recovered if one of the elements in that triple is complete.*

Proof. We assume that the data stored in the IPFS and the Hyperledger Fabric are permanent since all the data are duplicated and distributed in these systems. First, we suppose log_i is complete, and the other two elements are lost. According to the Definition 2 the *Log Provider* can regenerate the CID_i' through the API supported by the IPFS and $CID_i' = CID_i$. Then, the *Log Provider* can query the Hyperledger Fabric to find the key-value pair in which the value is equal to CID_i and the $TxID_i$ is found since the key is equal to "CID" + $TxID_i$ in that pair. Second, we suppose CID_i is complete, and the other two elements are lost. According to [9], we know all the files stored in the IPFS are content-addressed, which means the file can be found in IPFS by providing its CID. Hence, the *Log Provider* can recover log_i by using the CID_i to query the IPFS. Then, the $TxID_i$ can be found by repeating the procedure in the first scenario. Finally, we suppose $TxID_i$ is complete. According to our system design, the *Log Provider* can create the specific key which is equal to "CID" + $TxID_i$ and query the Hyperledger Fabric to find the value which is CID_i. Then, the *Log Provider* can recover log_i by following the step in the second scenario. Therefore, the lost data in the *Log Provider* local database can be recovered as long as one of the elements in that triple is complete. □

Meanwhile, the concern of single-point failure in the *Logger* and the *Auditor* can be resolved since there are multiple endorsers in each role to execute the program. These two roles can tolerate the point failure as long as the majority of endorsers are online and operating normally.

4.7 Conclusions

In this paper, we propose a blockchain-based audit log system to ensure log file data integrity. Our system can detect the log tampering attack from the log provider and logger, and reliably tolerate one-third of colluded nodes. We propose an NFT-based integrity proof generation method, which resolves the blockchain limited scalability problem by only storing small-sized proofs (sub-NFT) on the blockchain. This method also eliminates the concern of single-point failure and data loss in a central database by outsourcing log files to IPFS and recording integrity proofs in the blockchain network. Finally, we implement a prototype of our system and analyze whether our system can ensure data integrity. The experimental results show our system is reliable to tolerate one-third of colluded nodes, and our proof generation method can save approximately 50% storage space for Hyperledger Fabric. Meanwhile, the security analysis proves that our system can ensure log file data integrity under the threat model. In future work, the system performance of the logging and auditing processes need further improvement.

Chapter 5

Conclusions and Suggestions for Future Research

5.1 Conclusion

Researchers are exploring blockchain technology broadly to collaborate with other applications beyond cryptocurrencies. Its chain data structure and the combination with other technologies bring decentralization, immutability, and other properties, which can improve the security and reliability of the application systems.

In the work of the E-voting system, we propose a Double Blockchain-based E-voting system, which is the first E-voting system that covers all the core requirements of a reliable system. Our system consists of two blockchains: private blockchain and public blockchain. Both blockchains record the same voting data, and only the private blockchain records the voter's personal information, and the data recorded in the private blockchain are not allowed to be accessed by anyone until the election is finished. Voters can query the voting data and supervise the election tallying process in the public blockchain. Therefore, the voters' personal information can be audited in the private blockchain, and their privacy can be protected. The correctness of the system process can be verified by querying and supervising data on the public blockchain. Besides, we use the linkable ring signature to authenticate and encrypt voters' identification and restrict voters from multiple voting. Meanwhile, we propose an on-chain and off-chain hybrid storage mechanism, which combines the off-chain database with the distributed ledger, to ensure data consistency between the private blockchain and the public blockchain. Finally, we implement a prototype to evaluate and analyze our system. The experiment results show our system has a better performance when the block size is 512 KB, and the hybrid storage mechanism has not become the bottleneck of our system performance. The security analysis shows that our system is reliable by satisfying all core requirements.

In the work of the audit log system, we propose a secure and reliable blockchain-based audit log system to ensure log file data integrity. We set a general threat model which assumes the logger and auditor are untrusted and the log provider is only trusted when it generates log files. Our system based on this threat model has multiple loggers and auditors nodes to operate system processes. These nodes install the smart contracts and follow the consensus algorithm to reliably tolerate one-third of collusion nodes. We also employ the AES algorithm to protect log data privacy from compromised loggers and auditors. For resolving data storage flaws, we propose an NFT-based integrity proof generation method to record integrity proof (sub-NFT) in blockchain and outsource related log files to IPFS. This method resolves the limited blockchain scalability problem by only storing small-sized proofs on the blockchain. It also eliminates the concern of single-point failure and data loss in a central database by outsourcing log files to a distributed file system (IPFS) and recording integrity proofs in the blockchain network. Finally, we implement a prototype of our system and analyze whether our system can ensure data integrity. The experimental results show our system is reliable to tolerate one-third of colluded nodes, and our proof generation method can save approximately 50% storage space for Hyperledger Fabric. Meanwhile, the security analysis proves that our system can ensure log file data integrity under the general threat model.

5.2 Suggestions for Future Research

In future work, we will focus on improving the system performance and security of our current work.

5.2.1 System Performance Improvement

In the work of the E-voting, the system performance is unsatisfactory due to equipment limitations and the prototype is implemented on a personal computer. We need to set up a new prototype in a distributed environment with multiple devices and evaluate the system performance by setting different parameters and configurations. Besides, Although the off-chain database has not become the bottleneck of the system performance in the current experiment, it can still be the bottleneck when the read and write speed reaches the maximum value. We still need to consider how to improve the hybrid storage scheme to reduce the impact on system performance and make it securer.

In the work of audit log system, the performance of the logging and auditing processes can be further improved by resetting the *BatchTimeout* parameter, expanding the bandwidth, and building a prototype in a distributed environment with multiple devices. In addition, the AES key length may become an influencing factor since, according to AES design, the longer key length takes more time and resources to process files, and makes the algorithm more secure. How to set a proper key length in this system is another research direction.

5.2.2 Robustness and Usability

An E-system is considered robust if it can function properly even when some voters misbehave or if the system experiences partial failure. This typically requires a distributed system that can support fault tolerance. In the future, we should consider the influence of voters' misbehavior on our E-voting system and optimize our system to tolerate these effects. Then, we should also consider improving system usability to attract more people to use our E-voting system. For example, the need to improve the accessibility for disabled individuals in our E-voting system.

5.2.3 More General Threat Model in Audit Log System

In the work of the audit log system, we can set a more general threat model that the *Log Provider* is trusted only when it generates lof files and launches the logging process for the first time. Then, we can upgrade the system to be more robust. In this threat model, the *Log Provider* can regenerate or tamper with the log file and gets its related CID and TxID by normally transacting with the *Logger*. Then, it can send the new triple to the *Court* and avoid penalty. One possible solution is to compare the time stamp of proof recorded in the blockchain and the time stamp written in the log file during the auditing process.

References

[1] Infura.

[2] Voatz mobile voting platform an overview: Security, identity, auditability, 2020.

[3] Rafael Accorsi. Log data as digital evidence: What secure logging protocols have to offer? In *Proc. Annual IEEE International Computer Software and Applications Conference (COMPSAC'09)*, volume 2, pages 398–403, Seattle, WA, Jul. 2009. IEEE.

[4] Ben Adida. Helios: Web-based open-audit voting. In *Proc. USENIX Security Symposium (USENIX Security'08)*, volume 17, pages 335–348. San Jose, CA, USA, Jul. 2008.

[5] Ashar Ahmad, Muhammad Saad, Mostafa Bassiouni, and Aziz Mohaisen. Towards blockchain-driven, secure and transparent audit logs. In *Proc. International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous'18)*, pages 443–448, New York, USA, Nov. 2018.

[6] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proc. the thirteenth EuroSys conference (EuroSys'18)*, number 30, pages 1–15, Porto, Portugal, Apr. 2018.

[7] Leonardo Aniello, Roberto Baldoni, Edoardo Gaetani, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. A prototype evaluation of a tamper-resistant high performance blockchain-based transaction log for a distributed database. In *Proc. European Dependable Computing Conference (EDCC'17)*, pages 151–154, Geneva, Switzerland, Sep. 2017.

[8] Asaph Azaria, Ariel Ekblaw, Thiago Vieira, and Andrew Lippman. Medrec: Using blockchain for medical data access and permission management. In *Proc. International Conference on Open and Big Data (OBD'16)*, pages 25–30. IEEE, Aug. 2016.

[9] Juan Benet. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.

[10] Léo Besançon, Catarina Ferreira Da Silva, and Parisa Ghodous. Towards blockchain interoperability: Improving video games data exchange. In *Proc. International Conference on Blockchain and Cryptocurrency (ICBC'19)*, pages 81–85, Seoul, Korea (South), Jul. 2019.

[11] Johannes Braun, Franziskus Kiefer, and Andreas Hülsing. Revocation and non-repudiation: when the first destroys the latter. In *Proc. European Public Key Infrastructure Workshop (EuroPKI'13)*, pages 31–46, Berlin, Heidelberg, 2013.

[12] Thanh Bui. Analysis of docker security. *arXiv:1501.02967*, 2015.

[13] Luigi Castaldo and Vincenzo Cinque. Blockchain-based logging for the cross-border exchange of ehealth data in europe. In *Proc. International ISCIS Security Workshop (Euro-CYBERSEC'18)*, pages 46–56, Cham, Jul. 2018.

[14] Mumin Cebe, Enes Erdin, Kemal Akkaya, Hidayet Aksu, and Selcuk Uluagac. Block4forensic: An integrated lightweight blockchain framework for forensics applications of connected vehicles. *IEEE Communications Magazine*, 56(10):50–57, Oct. 2018.

[15] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[16] Boyuan Chen and Zhen Ming (Jack) Jiang. A survey of software log instrumentation. *ACM Comput. Surv.*, 54(4):1–34, May 2021.

[17] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On scaling decentralized blockchains. In *Proc. International Conference on Financial Cryptography and Data Security (FC'16)*, pages 106–125, Berlin, Heidelberg, Aug. 2016.

[18] Jordi Cucurull and Jordi Puiggalí. Distributed immutabilization of secure logs. In *Proc. International Workshop on Security and Trust Management (STM'16)*, volume 9871, pages 122–137, Sep. 2016.

[19] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael. 1999.

[20] Gaby G Dagher, Praneeth Babu Marella, Matea Milojkovic, and Jordan Mohler. Broncovote: Secure voting system using ethereum's blockchain. In *Proc. International Conference on Information Systems Security and Privacy (ICISSP'18)*, pages 96–107, Funchal, Madeira, Portugal, Jan. 2018.

[21] Prerit Datta, Natalie Lodinger, Akbar Siami Namin, and Keith S Jones. Cyber-attack consequence prediction. *arXiv preprint arXiv:2012.00648*, 2020.

[22] Alexander DeLaRosa. Log monitoring: not the ugly sister. *Pandora FMS. Archived from the original on February*, 14, 2018.

[23] Omar Dib, Kei-Leo Brousmiche, Antoine Durand, Eric Thea, and Elyes Ben Hamida. Consortium blockchains: Overview, applications and challenges. *International Journal On Advances in Telecommunications*, 11(1&2):51–64, 2018.

[24] Razvi Doomun, Jayramsingh Doma, and Sundeep Tengur. Aes-cbc software execution optimization. In *Proc. International Symposium on Information Technology (ITCC'08)*, volume 1, pages 1–8, Kuala Lumpur, Malaysia, Aug. 2008.

[25] Jordi Barrati Esteve, Ben Goldsmith, and John Turner. International experience with e-voting. *Norwegian E-Vote Project. International Foundation for Electoral Systems. Document disponibil online la adresa http://www. ifes. org/Content/Publications/News-in-Brief/2012/June/%7E/media/B7FB434187E943C18F4D4992A4EF75DA. pdf*, 2012.

[26] Joshua AT Fairfield. Tokenized: The law of non-fungible tokens and unique digital property. *Ind. LJ*, 97:1261, 2022.

[27] Yu Fei, Jing Ning, and Qing Hu. A log storage system based on block chain (in chinese). *Cyberspace Security*, 9(6):6, 2018.

[28] Massimo Franceschet, Giovanni Colavizza, Blake Finucane, Martin Lukas Ostachowski, Sergio Scalet, Jonathan Perkins, James Morgan, Sebástian Hernández, et al. Crypto art: A decentralized view. *Leonardo*, 54(4):402–405, Aug. 2021.

[29] J Paul Gibson, Robert Krimmer, Vanessa Teague, and Julia Pomares. A review of e-voting: the past, present and future. *Annals of Telecommunications*, 71(7):279–286, Jun. 2016.

[30] Nicole J Goodman and Jon H Pammett. The patchwork of internet voting in canada. In *2014 6th International Conference on Electronic Voting: Verifying the Vote (EVOTE)*, pages 1–6. IEEE, 2014.

[31] Rifa Hanifatunnisa and Budi Rahardjo. Blockchain based e-voting recording system design. In *Proc. International Conference on Telecommunication Systems Services and Applications (TSSA'17)*, pages 1–6, Lombok, Indonesia, Oct. 2017.

[32] Freya Sheer Hardwick, Apostolos Gioulis, Raja Naeem Akram, and Konstantinos Markantonakis. E-voting with blockchain: An e-voting protocol with decentralisation and voter privacy. In *Proc. International Conference on Internet of Things (iThings'18) and IEEE Green Computing and Communications (GreenCom'18) and IEEE Cyber, Physical and Social Computing (CPSCom'18) and IEEE Smart Data (SmartData'18)*, pages 1561–1567, Halifax, NS, Canada, Jul-Aug. 2018.

[33] Richard L Hasen. The 2016 us voting wars: From bad to worse. *Wm. & Mary Bill Rts. J.*, 26:629, Mar. 2017.

[34] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R. Lyu. A survey on automated log analysis for reliability engineering. *ACM Comput. Surv.*, 54(6):1–37, Jul. 2021.

[35] Friðrik Þ Hjálmarsson, Gunnlaugur K Hreiðarsson, Mohammad Hamdaqa, and Gísli Hjálmtỳsson. Blockchain-based e-voting system. In *Proc. International Conference on Cloud Computing (CLOUD'18)*, pages 983–986, San Francisco, CA, USA, Jul. 2018.

[36] Jen-Ho Hsiao, Raylin Tso, Chien-Ming Chen, and Mu-En Wu. Decentralized e-voting systems based on the blockchain technology. In *Proc. Advances in Computer Science and Ubiquitous Computing (CUTR'17)*, volume 474, pages 305–309, Singapore, Dec. 2017.

[37] Jiansen Huang, Hui Li, and Jiyang Zhang. Blockchain based log system. In *Proc. International Conference on Big Data (Big Data'18)*, pages 3033–3038, San Francisco, CA, USA, Sep. 2018.

[38] Lv Jiangfu, Yinggxu Lai, and Jing Liu. Log security storage and retrieval based on combination of on-chain and of-chain (in chinese). *Computer Science*, 47(3):6, 2020.

[39] Kashif Mehboob Khan, Junaid Arshad, and Muhammad Mubashir Khan. Secure digital voting system based on blockchain technology. *International Journal of Electronic Government Research (IJEGR'18)*, 14(1):53–62, 2018.

[40] Peter Lam. From helios to voatz: Blockchain voting and the vulnerabilities it opens for the future.

[41] Max Landauer, Florian Skopik, Markus Wurzenberger, and Andreas Rauber. System log clustering approaches for cyber security applications: A survey. *Computers & Security*, 92:101739, May 2020.

[42] Arjen K. Lenstra. Unbelievable security matching aes security using public key systems. In *Proc. International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT'01)*, pages 67–86, Berlin, Heidelberg, Nov. 2001.

[43] Xueping Liang, Sachin Shetty, Deepak Tosh, Charles Kamhoua, Kevin Kwiat, and Laurent Njilla. Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability. In *Proc. International Symposium on Cluster, Cloud and Grid Computing (CCGRID'17)*, pages 468–477, Madrid, Spain, Jul. 2017.

[44] Yi Liu and Qi Wang. An e-voting protocol based on blockchain. Cryptology ePrint Archive, Report 2017/1043, 2017. `https://ia.cr/2017/1043`.

[45] Zhuotao Liu, Yangxi Xiang, Jian Shi, Peng Gao, Haoyu Wang, Xusheng Xiao, Bihan Wen, and Yih-Chun Hu. Hyperservice: Interoperability and programmability across heterogeneous blockchains. In *Proc. ACM SIGSAC Conference on Computer and Communications Security (CCS'19)*, pages 549–566, London, United Kingdom, Nov. 2019.

[46] Ülle Madise and Tarvi Martens. E-voting in estonia 2005. the first practice of country-wide binding internet voting in the world. In *Electronic Voting 2006–*

2nd International Workshop, Co-organized by Council of Europe, ESF TED, IFIP WG 8.6 and E-Voting. CC. Gesellschaft für Informatik eV, 2006.

[47] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *Proc. IEEE Symposium on Security and Privacy (S&P'13)*, pages 397–411, Berkeley, CA, USA, May 2013.

[48] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, Oct. 2008.

[49] Truc Nguyen and My T Thai. zvote: A blockchain-based privacy-preserving platform for remote e-voting. In *Proc. International Conference on Communications (ICC'22)*, pages 4745–4750, Seoul, South Korea, May 2022.

[50] Shen Noether. Ring signature confidential transactions for monero. *IACR Cryptol. ePrint Arch.*, 2015:1098, 2015.

[51] William Pourmajidi and Andriy Miranskyy. Logchain: Blockchain-assisted log storage. In *Proc. International Conference on Cloud Computing (CLOUD'18)*, pages 978–982, San Francisco, CA, USA, 2018.

[52] Benedikt Putz, Florian Menges, and Günther Pernul. A secure and auditable logging infrastructure based on a permissioned blockchain. *Computers & Security*, 87:101602, Nov 2019.

[53] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, Feb. 1978.

[54] Reza Samavi and Mariano P. Consens. Publishing l2tap logs to facilitate transparency and accountability. *LDOW*, 2014.

[55] Bruce Schneier and John Kelsey. Secure audit logs to support computer forensics. *ACM Trans. Inf. Syst. Secur.*, 2(2):159–176, May 1999.

[56] Qun Song, Yuhao Chen, Yan Zhong, Kun Lan, Simon Fong, and Rui Tang. A supply-chain system framework based on internet of things using blockchain technology. *ACM Transactions on Internet Technology (TOIT'21)*, 21(1):1–24, Jan 2021.

[57] Michael A Specter, James Koppel, and Daniel Weitzner. The ballot is busted before the blockchain: A security analysis of voatz, the first internet voting application used in us federal elections. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1535–1553. USENIX Association, Aug. 2020.

[58] Shi-Feng Sun, Man Ho Au, Joseph K Liu, and Tsz Hon Yuen. Ringct 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero. In *Proc. European Symposium on Research in Computer Security (ESORICS'17)*, pages 456–474, Cham, 2017.

[59] Andrew Sutton and Reza Samavi. Blockchain enabled privacy audit logs. In *Proc. International Semantic Web Conference (ISWC'17)*, pages 645–660, Cham, Oct. 2017.

[60] Pawel Szalachowski, Stephanos Matsumoto, and Adrian Perrig. Policert: Secure and flexible tls certificate management. In *Proc. ACM Conference on Computer and Communications Security (CCS'14)*, page 406–417, Scottsdale, Arizona, USA, Nov. 2014.

[61] Baocheng Wang, Jiawei Sun, Yunhua He, Dandan Pang, and Ningxiao Lu. Large-scale election based on blockchain. *Procedia Computer Science*, 129:234–237, Mar. 2018.

[62] King-Hang Wang, Subrota K Mondal, Ki Chan, and Xiaoheng Xie. A review of contemporary e-voting: Requirements, technology, systems and usability. *Data Science and Pattern Recognition*, 1(1):31–47, 2017.

[63] Xu Wang, Guangsheng Yu, Ren Ping Liu, Jian Zhang, Qiang Wu, Steven Su, Ying He, Zongjian Zhang, Litao Yu, Taoping Liu, et al. Blockchain-enabled fish provenance and quality tracking system. *IEEE Internet of Things Journal*, 9(11):8130–8142, Sep. 2021.

[64] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[65] Haibo Yi. Securing e-voting based on blockchain in p2p network. *EURASIP Journal on Wireless Communications and Networking*, 2019(1):1–9, May 2019.

[66] Haoran Yuan, Xiaofeng Chen, Jianfeng Wang, Jiaming Yuan, Hongyang Yan, and Willy Susilo. Blockchain-based public auditing and secure deduplication with fair arbitration. *Information Sciences*, 541:409–425, Dec. 2020.

[67] Wenbin Zhang, Yuan Yuan, Yanyan Hu, Shaohua Huang, Shengjiao Cao, Anuj Chopra, and Sheng Huang. A privacy-preserving voting protocol on blockchain. In *Proc. International Conference on Cloud Computing (CLOUD'18)*, pages 401–408. San Francisco, CA, USA, Jul. 2018.

[68] Yuan Zhang, Chunxiang Xu, Nan Cheng, Hongwei Li, Haomiao Yang, and Xuemin Shen. Chronos $^+$: An accurate blockchain-based time-stamping scheme for cloud storage. *IEEE Transactions on Services Computing*, 13(2):216–229, Oct. 2020.

[69] Fei Zhu, Wei Wu, Yuexin Zhang, and Xiaofeng Chen. Privacy-preserving authentication for general directed graphs in industrial iot. *Information Sciences*, 502:218–228, Oct. 2019.