A Hybrid Blockchain-Based Event Ticketing System

A Thesis Submitted to the
College of Graduate and Postdoctoral Studies
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Mengxuan Liu

PERMISSION TO USE

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying, publication, or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head
Department of Computer Science
176 Thorvaldson Building
110 Science Place
Saskatoon SK S7N 5C9
Canada

OR

Dean
College of Graduate and Postdoctoral Studies
University of Saskatchewan
116 - 110 Science Place
Saskatoon SK S7N 5C9
Canada

ABSTRACT

Event ticketing systems are facing challenges of preventing ticket forgery and scalping while assuring privacy protection and information transparency. To alleviate these issues, this thesis presents a hybrid blockchain-based event ticketing system. It uses blockchain technology to ensure the transparency of ticketing information, and uses asymmetric encryption technology to protect privacy. The system also uses digital signature technology to ensure ticket authenticity, and has a novel ticket verification mechanism for preventing ticket scalping. A description of the experiments that were conducted on the system implementation and the analysis of their results conclude an evaluation of the system.

ACKNOWLEDGEMENTS

This thesis would not have been possible without the help and support of numerous individuals. First and foremost, I would like to give my cordial thanks to my supervising professor Prof. Ralph Deters, who has generously provided me with his guidance and help. He is the best mentor and advisor that any graduate student could hope for. He introduced me into the field of blockchain, led me all the way to the cut-edge frontier of modern technology, and trained me to become a better thinker and a researcher. Without his insightful comments and advice, this thesis would not have been accomplished to the level that I have felt so proud of.

My great thanks also go to Prof. Chris Zhang who helped and supported me all the way through my academic study, Prof. Li Zong who provided me with financial support and other help, Dr. Murray Bentham who reviewed my draft thesis word-by-word and provided a great many advice, and Prof. Julita Vassileva who not only taught me “Social Computing” course but also offered valuable advice on the improvement of the thesis.

I am extremely indebted to all the Defense Committee members, Prof. Julita Vassileva, Prof. Wahid Khan, Assistant Prof. Zadia Codabux, for their insightful and valuable comments and suggestions for the improvement of the thesis.

I would also like to extend my great gratitude to the great help and support from the SKSIS Project Group, Prof. Angela K. Bedard-Haughn, Dr. Murray Bentham, and other members for their support and advice on the completion of the thesis.

Last but not the least, my heartfelt gratitude goes to my parents and my family, my peer graduate students in the Madmuc Lab, Steven Xue, Heidong Wang, Yalin Chen, and others who helped me a lot in my study and research work, and all my friends who have given me encouragement and support.

CONTENTS

LIST OF TABLES

Table Number **Page Number**

LIST OF FIGURES

Figure Number **Page Number**

LIST OF ABBREVIATIONS

ACL	**Access Control List**
ASIC	**Application-Specific Integrated Circuit**
BFT	**Byzantine Fault Tolerance**
CFT	**Crash Fault Tolerance**
DLT	**Distribute Ledger Technology**
DPoS	**Delegated Proof of Stake**
EOA	**External Owned Account**
EVM	**Ethereum Virtual Machine**
GUID	**Globally Unique Identifier**
IPFS	**InterPlanetary File System**
MSP	**Membership Service Provider**
NIZK	**Non-Interactive Zero-Knowledge Proof**
PKI	**Public Key Infrastructure**
PBFT	**Practical Byzantine Fault Tolerance**
PoS	**Proof of Stake**
TPS	**Transactions Per Second**
UTXO	**Unspent Transaction Output**

CHAPTER 1: INTRODUCTION

The increasing use of electronic ticketing systems in recent years has greatly improved the efficiency of ticketing, management, and reduced the cost of storage of the ticketing information, compared to traditional paper tickets (Qteishat et al., 2014). However, there are still many challenges. Centralized ticketing systems lack transparency, the protection of consumer rights, the security and privacy of data stored, and traceable legers. To solve these problems, blockchain technology is used to provide decentralized solutions for developing event ticketing systems (Tackmann, 2017). Nevertheless, decentralized event ticketing systems have their share of problems too including efficiency, ticket scalping and ensuring privacy and security of data. This thesis presents a hybrid blockchain-based event ticketing system to solve these problems while ensuring transparency and traceability of the ticketing information. This thesis only focuses on event ticketing system such as concerts, performances, and other large-audience presentations. Single-user type ticketing systems such as flight, train, bus, rapid transport and movie ticketing will not be discussed.

Electronic event ticketing systems have not only enhanced the convenience of buying and selling tickets but also greatly eased issues of ticket management and loss (Qteishat et al., 2014). Nevertheless, problems exist in current ticketing systems. First, centralized systems have to be sufficiently robust to provide the necessary client trust. In centralized systems, a center controller responsible for processing, storing data and providing service, is usually supplied by trusted third parties (Nakamoto, 2008). These characteristics also apply to centralized electronic ticketing systems. For example, from the perspective of an event organizer, they only know the ticket sales information from the company responsible for sale. It is difficult for them to verify if the sale information is true or not. The same issue also applies to the consumers. The consumers cannot confirm how the ticket purchase information is stored. Hence, it is difficult for the consumers to verify if the tickets they bought are valid. Clearly, the centralized system requires the trust of all participants in the party playing the role of center controller. Second, privacy protection is an important issue for ticketing systems. For instance, consumers' records are useful for researching consumer behavior. But, the consumers' records are private and any use of their personal records should be authorized by the consumer. However, since the centralized system is a black

box to the consumers, it is difficult for them to establish trust. The security of ticketing systems is another important issue. In a centralized system, all the data is stored, maintained and processed in one place. Compared with a decentralized system, it is more vulnerable. The tampering of data is difficult to discover in a centralized system. In addition, the resale of tickets is a major problem faced by the current ticketing systems as the tickets can be resold at several times the original price. This kind of reselling behavior has caused confusion in the ticket market as no one benefits in the reselling except the middle ticket sellers.

To solve these problems, the decentralized solutions have been introduced in recent years (Aventus Protocal Foundation, 2018; Cha et al., 2018; GET Foundation Team, 2017; Hao, 2017; Ko et al., 2020; Lin et al., 2019). These solutions solved some of the ticketing systems' problems by using blockchain technology. However, problems still exist that have not been addressed in the research literature. This thesis reviews the research of blockchain-based event ticketing systems and discusses the problems not covered in the research literature.

As discussed previously, a ticket system should consider a variety of problems including ticket authenticity, privacy protection, system transparency, ticket traceability and scalping prevention. This thesis addresses these problems in the following method.

Ticket authenticity is an important problem for all ticketing systems (Finžgar & Trebar, 2011). Fake tickets can cause huge losses to event ticket sales. Therefore, how to ensure that the tickets cannot be counterfeited and the consumers can easily verify the authenticity and validity of their tickets needs to be explored.

There are a number of facets to privacy protection (Cha et al., 2018). The consumer's ticket buying records are considered to be private data. An event's ticket selling records are the privacy of the event organizers. However, some of the consumer's private data are required to be stored in a ticketing system. Hence, how to protect private data without affecting the operation of the ticketing system needs to be discussed and resolved.

Transparency is important to a ticketing system (Tackmann, 2017). Public information, such as the basic information of an event and ticketing agreement, should be unforgeable,

valid and accessible to the public. There needs to be a secure mechanism to ensure that public information and any changes to it are publicly supervised to prevent it from being hidden or tampered with.

Traceability is a basic requirement for any information system (Tackmann, 2017). In terms of ticketing systems, all the ticket related information such as the signed ticket agreement and ticket revocation proof, should be traceable. These records are valuable with many uses, for example, reconciling disputes.

There is much money to be benefited by illegally reselling tickets and therefore ticket scalping has become an inevitable problem that every ticketing system has to resolve. Ticket scalping can have a serious unexpected impact on normal ticket sales (Bell, 2005). Therefore, scalping prevention is one of the most important factors that must be seriously considered in the design of a ticketing system.

- **Outline**

This dissertation is comprised of six chapters. The first chapter is a brief introduction of the study's background. Chapter 2 introduces the current status of development of event ticketing systems and the research problems requiring attention and solution. Chapter 3 reviews and discusses the blockchain technology literature including related technologies, and offers analysis of existing decentralized solutions for event ticketing systems. Chapter 4 discusses the architecture of the Hybrid Blockchain-Based Event Ticketing System, explaining how the research problems are solved, and describes a conceptual model of the infrastructure as well as time sequence of the major functions, to illustrate the working principle of the system. In Chapter 5, the evaluations and experiments conducted on the implementation of the solutions of the system are discussed as related to the research problems. Finally, Chapter 6 presents a conclusion and introduces future research work that could be investigated.

CHAPTER 2: PROBLEM DEFINITION

This Chapter discusses the problems commonly existing in event ticketing systems, illustrated in Figure 2.1.

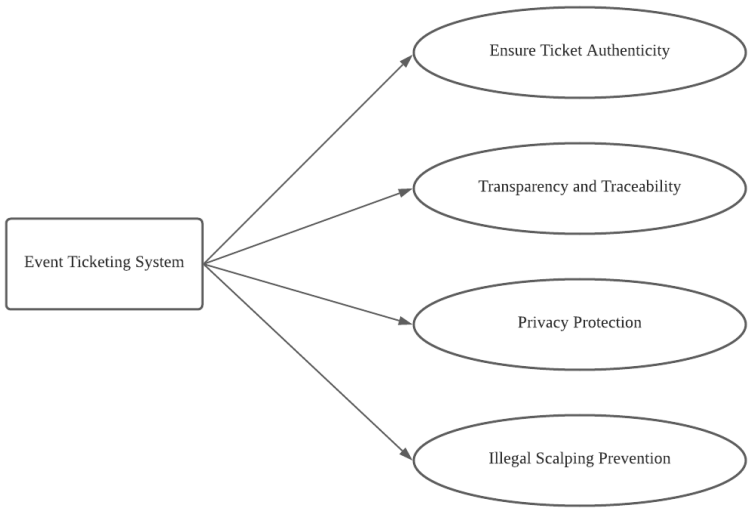


Figure 2.1 Problem Definition

The purpose of this thesis is to develop an event ticketing system architecture that addresses the problems illustrated in Figure 2.1. The objectives are to provide secure ticketing, making tickets unforgeable, secure data storage, protecting users' (including consumers and event organizers) privacy while maintaining the necessary transparency to different user groups, and a ticket verification mechanism that prevents scalping. To achieve these goals, the following methodologies will be employed to address the problems.

1. How to build a ticketing system that can produce unforgeable tickets?
   For any ticketing system, fake tickets are always the most common issue and most important problem to solve. The conceptual idea is to produce traceable tickets with unforgeable evidence to prove the ticket is genuine. The user can query their ticket information and verify its authenticity from the evidence.

2. How to ensure transparency and traceability of data to protect consumer rights?
   Ticketing transparency and the traceability of the ticket records serve to protect users' rights and interests from being violated. Centralized ticketing systems are

perceived as a black box, prohibiting the ticketing system to be supervised by all the participants except for the center controller. Public information, including event time, location, ticket and revocation agreements, and similar information, should be supervised and traceable by both consumers and event personnel. This ensures that public ticket information cannot be unilaterally modified and thereby secures the public information records which can be used as evidence to protect consumers' rights and interests when there is a dispute about the public information. Similarly, ticket sales should also be supervised by all the event organizers. Blockchain technology is a suitable problem solution since each blockchain-recorded transaction is stored and processed by all nodes. Furthermore, the blockchain ensures that these transactions are traceable and tamper proof.

3. How to protect the users' privacy?

   As discussed in question 2, although the transparency of ticketing system information is very important for the protection of users' rights, equally important is the protection of user privacy. In a ticketing system, the consumer's real world identity should be considered and protected as the consumer's privacy. Similarly, the event's ticket sales information should be the privacy of the event's organizers. To solve this privacy problem, while maintaining transparency, the users should be divided into two categories, consumers and event participants because transparency and privacy are different for both these participants.

   For consumers, the requirement is for the ticketing system to not only safeguard their real world identity but also prohibit their ticketing activities from being traced, while all the time keeping the ticket information transparent and traceable by the consumer. Encrypting ticket and purchaser identity information can solve the problem of protecting the privacy of ticket consumers in a transparent ticketing system.

   For event participants, the need is for private event data, such as event ticket sales information, to be only shared among all the event participants and not be accessible to the outside world.

The solution to this problem lies in using different types of blockchain technology to meet the different types of requirements.

4. How to prevent ticket scalping?

   Ticket scalping (ticket reselling) is always a problem in event ticketing systems. Illegal ticket reselling can create huge chaos to an event and the ticket market in general. It can be prevented by a validated ticket verification procedure before audience entering. Requiring the use of valid documents, such as ID cards, driver's license or face ID, to purchase a ticket, is an effective way to prevent scalping. Nonetheless, these methods link sold tickets to consumers' real world identities, and makes their purchase records traceable, hence, consumers' privacy has failed. Therefore, a new method should be explored to prevent ticket scalping while maintaining consumer privacy.

CHAPTER 3: LITERATURE REVIEW

This chapter reviews the literature to introduce blockchain and other related technologies that are used to solve the blockchain-based event ticketing system problems, while also discussing and analyzing other solutions too.

3.1 Blockchain

This section reviews the origin of the blockchain technology, the working principle of the Proof of Work (Nakamoto, 2008) which is the consensus algorithm of the Bitcoin blockchain, the work flow of the Bitcoin blockchain network (Nakamoto, 2008), the comparison of the different blockchain types, the comparison of different blockchain platforms (Ethereum and Hyperledger Fabric), and gives a discussion about the previous researches on blockchain-based event ticketing system solutions.

3.1.1 Origin of Blockchain

The concept of blockchain technology was originally introduced by Nakamoto Satoshi in a paper titled “Bitcoin: A Peer-to-Peer Electronic Cash System” in 2008 (Nakamoto, 2008). Nakamoto presented a novel peer-to-peer network system to solve the problem of double-spending in a decentralized peer-to-peer electronic cash system where payment transactions do not need to be verified by a third party. In the network, a submitted transaction is verified before being accepted by the network. The network verifies whether the submitted transaction has already been spent based on the history of all the transactions. All the verified transactions within a period of time are hashed into a Merkle Tree (Becker, 2008) to generate a new block. In the same transaction block and for transactions from the same payer, only the first-in transaction will be accepted in order to avoid double spending. Each block contains a hash of the previous block as a clue in forming a chain of blocks. A new generated block is broadcast to all the nodes in the network for verification, and the verified block will be added into a growing chain of blocks. Each node in the network always accepts the longest verified chain received and attempts to generate a new block in the longest chain. As a result, once a transaction has been recorded in a block in the chain, it cannot be changed. If an attacker wants to forge a transaction, the attacker must regenerate a block containing the fabricated transaction and all the blocks after it to catch

up and become the longest chain. This requires the attacker to have the majority of the computing power in the network, because the nodes in the network generate blocks through a proof-of-work mechanism which is based on computing power. In order to avoid this situation, Nakamoto (2008) also designed an incentive mechanism in his network system. A node that generates a new block will receive a new coin as a reward. In this way, if an attacker has the majority of the network's computing power to ensure that they can always generate a new block faster, then all the coins produced next in the network will be owned by this attacker. Therefore, the attacker will get more benefits by keeping honest.

3.1.2 Proof of work

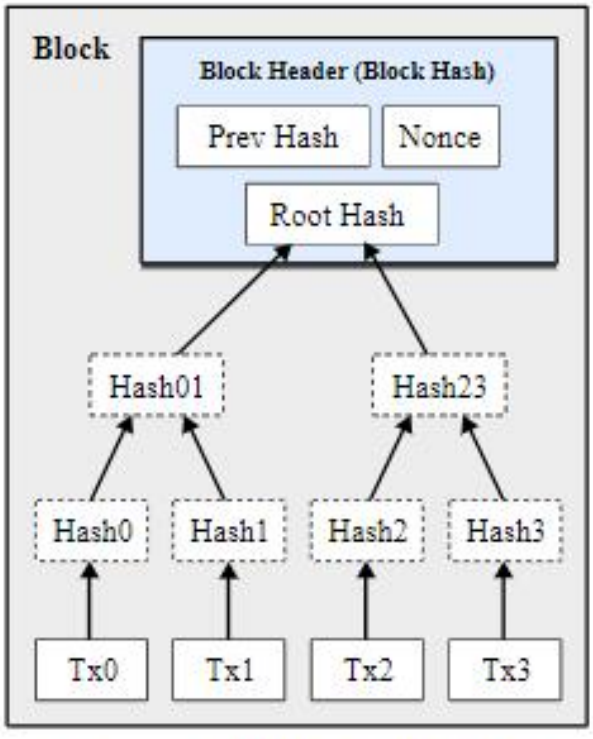
In "Bitcoin: A Peer-to-Peer Electronic Cash System" (Nakamoto, 2008), the author uses a proof-of-work mechanism to ensure that the decision making of a new block generation is based on computing power. This section reviews the implementation of the proof-of-work mechanism and the composition of a block given in Bitcoin blockchain.

The idea of making consensus among a network by solving a difficult, computing power consuming problem, was first introduced in 1998 in the white paper b-money by Dai, W (1998) but unfortunately a detailed description of the implementation was not included.. The author's idea was implemented by using the method of solving a difficult hash puzzle to reach consensus in a network by Finney, H (2004). But the method used by Finney, H (2004) was implemented in a centralized system.

The methodology used in Bitcoin blockchain to implement the proof-of-work mechanism is that when a node attempts to generate a new block, it has to constantly generate nonce until it generates a nonce which can make the hash value of the block (composed by the previous block hash, the root hash of the Merkle tree (Becker, 2008) of transactions in the block, and the nonce) starting with a certain number of zero bits. In this method, the node with more computing power, has faster processing speed, which will allow it to have higher probability to be the first one finding the nonce that meets the condition. After a node finds a nonce that meets the requirement, it will send the found nonce with the generated new block to the other nodes in the network. The other nodes will verify whether the nonce

meets the condition when they receive a new block. If the requirement is met, they will stop working on the current block generation and add the received new block to the current longest chain they have. After this, they will start to attempt to find a new nonce that meets the condition to generate a new block based on the lengthened chain. The interval of the generation of a new block can be controlled by the difficulty of proof-of-work. The more digits of zero bits required at the beginning of the hash value of the nonce, the greater the difficulty of the proof-of-work, and thereby it will take longer for nodes to find the nonce to generate a new block. With this mechanism, once a transaction is recorded in a block on the chain, it cannot be changed. If an attacker wants to change a previous transaction, they must regenerate the block where the transaction resides and all the subsequent blocks until the forged chain is longer than the current chain, which requires the attacker to have more computing power than all the other nodes combined. The process of generating a new block is also called mining (Chung et al., 2019).

According to Nakamoto (2008), the composition of a block in Bitcoin Blockchain is shown in Figure 3.1.



Transactions Hashed in a Merkle Tree

Figure 3.1 Block Composition of Bitcoin Blockchain (Nakamoto, 2008)

As Figure 3.1 illustrates, a block in the Bitcoin Blockchain has a Block Header and a Merkle Tree (Becker, 2008) formed by transactions. The Block Header is composed of three elements, Previous Hash, nonce, and Root Hash of the Merkle Tree. The Previous Hash is a hash value of the previous block which is used as a clue to link a block to the previous block. The nonce is a found nonce, meeting the condition of proof-of-work. The Root Hash is a Merkle Tree's root. The Merkle Tree in a block is formed by all the transactions recorded in that block.

3.1.3 Work Flow of the Bitcoin Blockchain Network

This section reviews the Work Flow of the Bitcoin Blockchain Network using Figure 3.2 Flowchart.

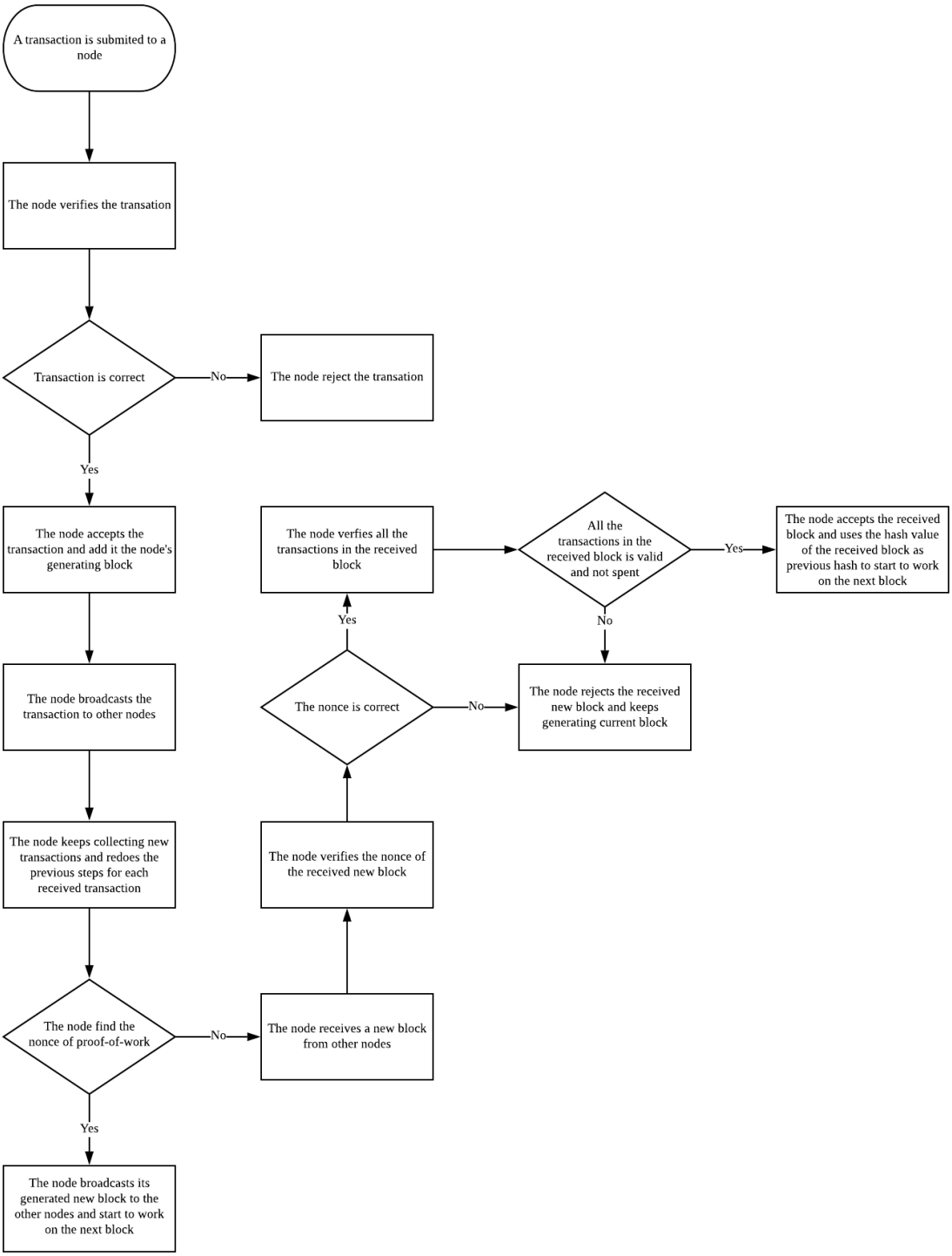


Figure 3.2 Flowchart of Bitcoin Blockchain Network Work Principle (Summarized from Nakamoto, 2008)

As can be seen in Figure 3.2, the work flow of the Bitcoin Blockchain Network can be described using the following steps.

Step 1: The work flow starts with a transaction being submitted to a node of the Bitcoin Blockchain.

Step 2: After a node receives the submitted transaction, the node (Node A) verifies if the transaction is valid and if the coins in it have been spent or not.

Step 3: If the submitted transaction passes the verification in Step 2, Node A will accept the transaction by adding it to the block Node A is generating. If not, Node A rejects the transaction.

Step 4: Node A broadcasts the verified transaction to other nodes in the network.

Step 5: Node A continues to collect new transactions and repeats the previous steps for each received transaction while working on finding the nonce that meets the condition of proof-of-work.

Step 6: If Node A finds a nonce that meets the condition of proof-of-work, it will generate a new block with the found nonce and all the verified transactions. Then it will broadcast the new generated block to the other nodes and start to work on generating a new block. If other nodes find the nonce faster, Node A will receive the generated new block broadcasted by other nodes.

Step 7: After Node A receives a new block from other nodes, it will first verify the nonce in the new block. If the nonce is correct, Node A will do the following steps. If not, Node A will reject the received block and keep working on generating its current block.

Step 8: After the nonce of the received block is verified, Node A will verify all the transactions inside the received block. If all the transactions are valid and have not been spent, Node A will do the next step. If not, Node A will reject the received block and keep working on generating its current block.

Step 9: After all the transactions in the received block are verified, Node A will accept the received block, and start to work on generating the next block based on the accepted block by using the hash value of the accepted block as the previous value in its generating block.

According to Figure 3.2 work flow, the nodes in the network will keep collecting new transactions to add to the generating block it is working on until it generates or accepts a new block. However, according to a publication by Göbel & Krzesinski (2017), in the actual Bitcoin blockchain, the size of a block is limited to 1MB and the number of transactions contained in a block cannot exceed 4000 transactions (Göbel & Krzesinski, 2017).

3.1.4 Types of Blockchain

In the previous sections, this thesis reviewed the origin of blockchain technology and the working principle of the Bitcoin Blockchain which is also the first blockchain. As discussed in Section 3.1.2, the goal of the proof-of-work in the Bitcoin Blockchain is to force all the nodes in the system to reach consensus on the distribution of data. In a distributed system, the method of reaching consensus among all the nodes is called consensus algorithm. Blockchain technology is considered to be a form of distributed ledger technology (DLT) which is also a distributed system (Wüst & Gervais, 2018). The proof-of-work is the consensus algorithm used in the Bitcoin blockchain. As new consensus algorithms are developed, new blockchain networks are introduced, following the original Bitcoin blockchain. According to Peck (2017), as well as Wüst and Gervais (2018), although different blockchains are based on a variety of consensus algorithms, they can be divided into two general types according to the rules of nodes joining the network. One type is the public blockchain, where anyone can choose to join the network by becoming a node to read and write transactions, as well as mining new blocks, or can exit the network at any time by abandoning its nodes. The other type is the permissioned blockchain, which only allows limited designated participants to join the network or become a node. Mingxiao et al, (2017) divided blockchains into three categories. In addition to the public blockchain and the permissioned blockchain, Mingxiao et al. (2017) also introduced a type of private blockchain which is a centralized system where an owner has the highest authority over all the data. However, Wüst and Gervais (2018) considered this type of blockchain as a special form of the permissioned blockchain.

Public Blockchain

Public Blockchain is also called permissionless blockchain, which is a totally decentralized peer-to-peer system. In a public blockchain, there is no trust required for any node. Every node has the same read and write authority as well as a complete ledger of all the past transactions on the blockchain, and all the nodes participate in the process of mining blocks. Similar to the original Bitcoin blockchain, most of the public blockchains reward the node that mined a new block with an incentive mechanism. The public blockchain consensus algorithm ensures the blockchain it works for meets the requirements of these features. The Bitcoin blockchain is a type of public blockchain.

Besides the proof-of-work of the Bitcoin blockchain, Mingxiao et al. (2017) reviewed several other public blockchain consensus algorithms, including Proof-of-Stake (PoS) introduced by King and Nadal (2012) and Delegated Proof-of-Stake (DPoS) introduced by Larimer (2014). The proof-of-stake is an algorithm that introduced a new concept of coin age based on the proof-of-work. The privileges of a coin will be increased over time from the time of its creation. The node which owns more coins with longer coin age will have more rights in the decision-making of the network. In this way, the waste of resources in proof-of-work will be reduced. The delegated proof-of-stake algorithm is a public blockchain algorithm with improved efficiency and reduced resource wasting, based on the proof-of-stake algorithm. In general, the consensus algorithms of the public blockchain must be able to allow nodes the freedom to join and leave the consensus progress in the network.

Different from Mingxiao et al. (2017), Peck (2017) and Wüst and Gervais (2018) discussed the public blockchain from the perspective of applicable scenarios. Based on the review of these two papers, the applicable scenarios and features of the public blockchain can be generalized as shown in the Figure 3.3.

Figure 3.3 Applicable Scenarios of Public Blockchain

As illustrated in Figure 3.3, according to Peck (2017) and Wüst and Gervais (2018), there are four features of the public blockchain and three applicable scenarios for choosing the public blockchain as the desired system. The features are:

Feature 1: The system does not have any limitations for the participants joining the system, or becoming nodes involved in reading and writing in the system, which usually results in public blockchains having a large number of nodes. This also ensures that public blockchains are not vulnerable under attacks.

Feature 2: There is no need for any trust among all the participants and nodes in the system.

It can be inferred from Feature 1 and Feature 2 that everyone can join, leave, become a node or abandon a node at any time in the system.

Feature 3: The efficiency of the system in processing transactions is low (using the number of transactions processed per second (TPS) as the evaluation standard). As mentioned, different public blockchains use various consensus algorithms. However, the efficiency of these algorithms in public blockchains is low due to the magnitude of the nodes, which in turn results in relatively low efficiency of the public blockchains compared to the centralized systems. As explained, centralized systems do not need to make consensus over the network. And the permissioned blockchains make consensus more efficient than the public blockchains since their smaller scale network of limited nodes. Therefore, it is not recommended to use the public blockchains for operations that require high efficiency.

Feature 4: The system is completely decentralized, which means that all the nodes have the same authority and the same operations.

Feature 5: Once the data is written into an accepted block in the system, it cannot be changed (including modification and removing), which ensures the traceability of the data in the system.

Based on these features, according to Peck (2017) and Wüst and Gervais (2018), the applicable scenarios of the public blockchain are as follows:

Scenario 1: There is no trusted third party in the target system that can be trusted by all the participants. Feature 2 and Feature 4 are corresponding to this scenario.

Scenario 2: The Data in the target system needs to be distributed and stored to ensure that it will not be lost or tampered under attacks. Feature 1 and Feature 5 are corresponding to this scenario.

Scenario 3: The data in the target system should be public, transparent and traceable to all the participants in the network (the data can be encrypted ciphertext). Feature 5 is corresponding to this scenario.

Permissioned Blockchain

The permissioned blockchain is a system that can be considered both centralized and decentralized according to different roles it plays. For the nodes in the network, the permissioned blockchain is a decentralized system whose decision-making is based on the

consensus of the nodes in the network instead of a central role that can decide everything. For the participants, other than nodes, the permissioned blockchain is a centralized system which is a black box with no transparency. The permissioned blockchain is composed of a limited small number of nodes. Every node in the network is authorized and known. Therefore, the nodes in the permissioned blockchain can reach consensus by direct voting, which is not feasible for the public blockchain where the nodes are unknown and limitless. If direct voting was adopted to reach consensus in the public blockchain, a malicious attacker could increase its possibility of tampering with data by having more nodes, without incurring a huge cost. This characteristic determines that the permissioned blockchain is suitable for a limited scale network with limited number of nodes instead of a global scale network. The limited scale and the nodes being known make the consensus algorithm of the permissioned blockchain more efficient than the public blockchain.

Mingxiao et al. (2017) reviewed a consensus algorithm widely used in the permissioned blockchain called Practical Byzantine fault tolerance (PBFT) which was introduced by Castro and Liskov (1999). The PBFT is developed based on the Byzantine Generals problem which was introduced by Lamport (1983). The Byzantine Generals problem described a problem that existed in a distributed system network whose nodes reach consensus by passing messages to each other. However, some of the nodes can be unreachable, and furthermore, there could be dishonest nodes sending tampered fake messages, all of which may result in failure for the network to reach a correct consensus to make the right decision. If a distributed system network can keep reaching consensus to make right decision in such a situation, then this system network is considered a Byzantine Fault Tolerance (BFT) system. In the PBFT, for a network with n nodes, a malicious attacker needs to control more than (n-1)/3 nodes in the network to be able to tamper with the data. The data will be secure and the network is BFT as long as (2n+1)/3 of the nodes in the network are reachable and honest. The nodes in the network are divided into two types. One type of the node can receive, reply to client requests, and initiate and participate in the consensus reaching process. The other node type can only participates in the consensus reaching process.

There are five stages for process of nodes in the network to reach consensus on accepting a client request which include request, pre-prepare, prepare, commit and reply.

In the request stage, a master node receives and timestamps a client request. Then, the master node initiates the consensus process by sending messages to the other nodes in the network to permit them to make a decision on whether to accept the broadcasted client request. This is the pre-prepare stage. In the prepare stage, if a node (including master node and the other nodes) makes a decision to accept the client request, it will broadcast a message of its acceptance to all the other nodes. When a master node receives the acceptance messages from more than (2n+1)/3 nodes (including itself), it will start the next stage. The nodes in the commit stage will broadcast a commit message to the other nodes in the network. Similar to the previous stage, after a master node receives the commit message from more than (2n+1)/3 nodes (including itself) in the network, the master node executes the client request for the network to accept and the next step begins once the execution is completed. In the reply stage, the master node in the network responds to the client with the result of the execution of the client request.

In addition, Mingxiao et al. (2017) also reviewed a consensus algorithm called RAFT introduced by Ongaro and Ousterhout (2017) suitable for the private blockchain which can be considered a special type of the permissioned blockchain. The RAFT (Reliable, Replicated, Redundant, and Fault-Tolerant) algorithm is designed based on the Paxos algorithm which was first introduced by Leslie (1998) and then introduced again by Lamport (2001). Similar to the PBFT, Paxos is also developed based on the Byzantine Generals problem. The difference is that Paxos is a Crash Fault Tolerance (CFT) algorithm instead of BFT. For a distributed system network, where some nodes may be unreachable, if the system can still reach a consensus, the system can be considered as a CFT system. Compared with BFT, CFT can only ensure that the system network can reach a consensus, but it cannot guarantee that the consensus reached is correct when there are dishonest nodes inside the network. In Paxos, all the nodes in the network are required to trust an elected leader node. Therefore, as long as the leader node is honest, the consensus reached by the network will be correct, which makes Paxos suitable for private blockchains. For example, if there are a total of n nodes in the network, as long as n/2 nodes are reachable, the network

is CFT, which means it can work normally to reach a consensus. However, Paxos is too complex and difficult to implement in the distributed systems. To solve this problem, RAFT was introduced as a more concise solution in 2013.

According to the characteristics of the permissioned blockchain, the applicable scenarios of the permissioned blockchain were introduced and discussed by Peck (2017) as well as Wüst and Gervais (2018), which can be generalized as shown in Figure 3.4.

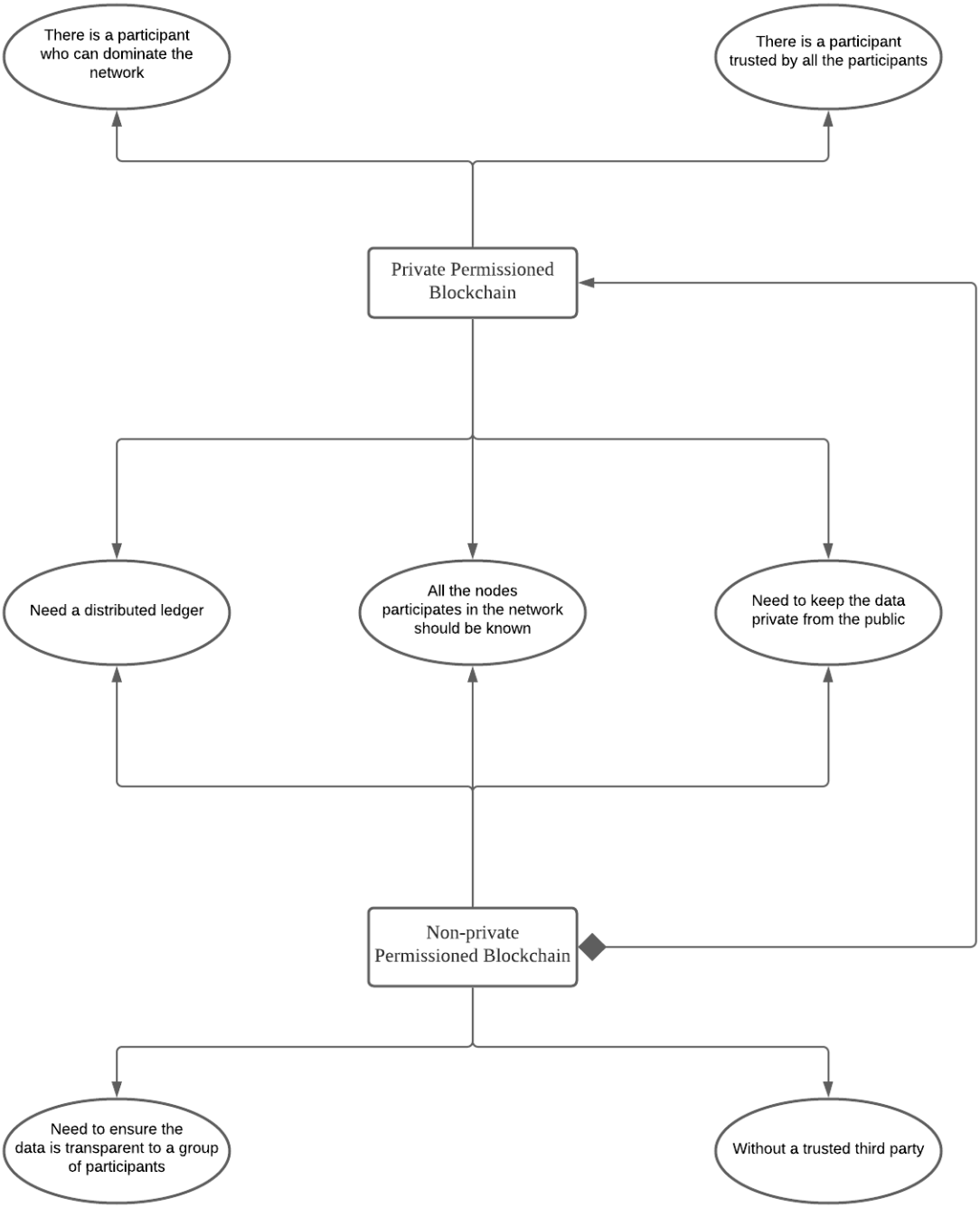


Figure 3.4 Applicable Scenarios of Permissioned Blockchain

As can be seen in Figure 3.4, the applicable scenarios of the permissioned blockchain discussed by Peck (2017) and Wüst and Gervais (2018) can be generalized as follows.

1. The data in the system needs to keep its privacy from the public. The system should be a black box to the public.

2. There are limitations for becoming a node in the system. Every node participating in the network should be known and authorized.

3. The system needs a distributed ledger to ensure the transparency or the security of the data (Private permissioned block uses the distributed leger to enhance the security of the data).

The above scenarios are applicable to all types of permissioned blockchain. According to Peck (2018), the permissioned blockchains can be categorized into two types, which are the non-private permissioned blockchain and private permissioned blockchain. In addition to the above three applicable scenarios for all types of permissioned blockchains, these two different types of permissioned blockchain have their own specific applicable scenarios.

For the non-private permissioned blockchain:

1. The system needs to ensure the transparency of the data among a group of participants.

2. There is no such third party that is trusted by all the participants in the network.

For the private permissioned blockchain:

1. There is a participant that is trusted by all the other participants.

2. There is a participant who has the highest authority that can dominate the network and dictate the authorization for the other nodes.

Comparison

A general comparison between different types of blockchain is concluded as shown in Table 3.1.

	Public Blockchain	Non-private Permissioned Blockchain	Private Permissioned Blockchain
Limitation of Becoming Nodes	No	Yes	Yes
Trust of Nodes	No Trust	Low Trust	High Trust
Scale of Participants	Global	Multiple (Limited)	Single
Transparency	Completely (Transparent to the Public)	Limited (Transparent to the Group)	Not Transparent
Trusted Third Party	No	No	Yes
Consensus Algorithms (Partial)	PoW, PoS, DPoS	PBFT	Paxos, RAFT
Throughput	Low	Medium	Medium High

Table 3.1 Comparison of the Different Types of the Blockchain

As shown in Table 3.1, different types of blockchain are compared based on the following characteristics summarized on previous work (Peck, 2017; Wüst and Gervais, 2018; Mingxiao et al., 2017).

1. The public blockchain has no limitation for nodes wishing to become part of the blockchain network. Everyone can choose to become a node of a public blockchain, while only limited authorized participants can become nodes in a permissioned blockchain (including non-private permissioned blockchain and private blockchain).

2. With regard to trustworthy nodes, there is no trust among all the nodes in a public blockchain network. In a non-private permissioned blockchain, each node and its participants must be known and authorized to join the network, but there is no need for trust among all the nodes in the network. In a private permissioned blockchain, not only does each node need to be authorized and known to join the network, but the nodes in the network also need to unconditionally trust a leader node and be dominated by it.

3. In terms of the scale of participants, a public blockchain participant is global, while a non-private permissioned blockchain has limited multiple participants, and a private permissioned blockchain has an owner.

4. As for transparency, a public blockchain ensures that it is transparent to the public (the data can be ciphertext). A non-private blockchain is transparent to all the nodes in its network, but it is a black box (not transparent) to the outside. And in a private permissioned blockchain, all the data is only transparent to a leader node which is owned by the owner of the network, and the leader node determines which data other nodes can access.

5. For the trusted third party, there is no such third party trusted by all the participants in either the public blockchain nor the non-private permissioned blockchain networks.

However, the owner of a private permissioned blockchain can be fully trusted by the network.

6. In the previous subsections, some of the consensus algorithms and their features have been introduced and reviewed. In review, the consensus algorithm in the original Bitcoin blockchain is based on Proof-of-Work, which is also a popular consensus algorithm used in many other public blockchains with different detailed implementations. Proof-of-Stake is a more efficient consensus algorithm in comparison with the Poof-of-Work, and it is gradually being applied to current public blockchains. Delegated Proof-of-Stake is an optimization consensus algorithm based on PoS, but is more efficient than PoS. In terms of the permissioned blockchain, most consensus algorithms are developed based on the Byzantine Generals Problem. The Practical Byzantine Fault Tolerance algorithm (PBFT) is used in the non-private permissioned blockchain while the Paxos is developed for the private permissioned blockchain. However, the Paxos is complex to implement. Therefore, the RAFT algorithm was developed based on the Paxos algorithm as a simpler implementation solution.

7. Throughput depends on the scale of the blockchain and the consensus algorithm used. Due to the global scale, the throughput of a public blockchain is low. In contrast, the nodes in a permissioned blockchain are known and limited. Therefore, although it cannot be compared with the speed of a centralized system, the nodes in a permissioned blockchain can reach consensus much faster than a public blockchain. The throughput of a permissioned blockchain depends mainly on the number of the nodes in the network. With the same number of nodes, the throughput of a private permissioned blockchain is greater than a non-private permissioned blockchain because its consensus algorithm is faster.

3.1.5 Blockchain Platforms

Since the first blockchain was introduced by Nakamoto Satoshi in 2008, the value of applying blockchain technology and cryptocurrency has gradually been discovered, as it provides a feasible solution for decentralized systems. As a result, a variety of new blockchain technologies were subsequently published in the following years and

blockchain-based platforms and models were introduced to fulfill the needs of blockchain-based application development. In the next section, two current major blockchain application development platforms are reviewed including Ethereum which is a public blockchain platform introduced by Vitalik (2013). The second platform reviewed is Hyperledger Fabric, a platform providing permissioned blockchain solutions introduced by Cachin (2016).

Ethereum

Vitalik Buterin is the founder of Ethereum, a public blockchain platform first introduced in 2013 in his introductory paper. Ethereum's goal is to provide developers simple and common solutions for building blockchain-based decentralized applications using smart contracts and decentralized application platforms. In 2016, Vatilik published a review paper of the Ethereum platform which provided a simplified explanation on the basic design.

According to Vatilik (2016), the infrastructure of the Ethereum platform is based on a public blockchain that uses a consensus algorithm called Ethash. Similar to the Bitcoin blockchain, the Ethash algorithm is based on the proof-of-work mechanism. However, the detailed working principle of the Ethash algorithm is different from the consensus algorithm used in the Bitcoin blockchain. As discussed previously, the Bitcoin blockchain's mining process of proof-of-work is based on CPU power by constantly calculating hard hash values (using SHA (Secure Hash Algorithm) (Eastlake & Jones, 2001)) until conditions are met. However, this mining process encourages the emergence of application-specific integrated circuit (ASIC). An ASIC is a special machine designed specifically for mining, which is believed not suitable for the Ethereum blockchain. Thus, Ethash is designed to use a method called memory hard hash function to implement its mining process of proof-of-work. Ethash evaluates the performance of node machines' memory, instead of CPU or GPU power to avoid the appearance of ASIC in the Ethereum network. Ethash also has an incentive mechanism called uncle incentivization, which is different from the incentive mechanism used by Bitcoin blockchain. This different process of mining resulted in a problem of adjustment mechanism of block mining and a transaction limitation in the Ethash. In addition, according to Vitalik (2016), one of the

biggest differences between the Ethereum blockchain and the Bitcoin blockchain is the composition of the Merkle tree (Becker, 2008) of a block. In the Bitcoin blockchain, the Merkle tree of a block only contains the hash values of the transactions included in that block. However, the Merkle tree of a block in the Ethereum blockchain contains all the transactions in the entire current state of the network, and thereby all the transactions in all the blocks of the network can be tracked by the Merkle Tree in the latest block. Buterin (2016) believed that compared to the Bitcoin block, this design makes the query of recorded transactions faster and easier.

In addition to the difference in the infrastructure design of the blockchain, the major advantage of the Ethereum platform is that it is built with an abstract functional layer running on top of the Ethereum blockchain. This is accomplished by building a stateful Turing-complete virtual machine called Ethereum Virtual Machine (EVM), which provides a solution for developers to create decentralized applications easily without rebuilding a new blockchain. Developers who use the Ethereum platform can only focus on developing conceptual models of their decentralized application in the abstract functional layer without knowing how their application is implemented in detail in the infrastructure level of the Ethereum blockchain. The EVM interprets abstract level smart contracts into chain-code that can be run in the Ethereum blockchain. Compared with the Ethereum blockchain, the Bitcoin blockchain does not support loops, although there are many script languages designed for the Bitcoin blockchain to implement a similar loop functionality. This is characteristic by the design of the Bitcoin blockchain so as to avoid infinite loops during the process of verifying transactions. Therefore, Ethereum platform is Turing-complete since the EVM makes it support loop functionality, while the Bitcoin is not Turing-complete.

Similar to Bitcoin, the Ethereum blockchain also produces its own cryptocurrency called Ether. Ether is not infinitely divisible. It has a smallest unit of value called Wei, named after Wei Dai (1998), the author of the B-money white paper. However, unlike the unspent transaction output (UTXO) based ledger which is used by the Bitcoin blockchain, the Ethereum blockchain is implemented with an account-based ledger. There are two types of accounts designed in the Ethereum platform. One is called External Owned Account (EOA)

which is used to store the users' balance of owned ether, and the other is contract type account used to execute code of smart contracts and store related data. The Ethereum platform uses the account types of the recipient to determine the purpose of transactions. Specifically, if the recipient of a transaction is an EOA type or unknown type account, the transaction will be executed as a transfer. If the recipient is a contract type account, the transaction will be executed using the code in the contract. The execution of contracts will cost computing power of the Ethereum platform. The computational effort invested in the execution of contracts is evaluated in a unit called gas which can be paid in ether.

According to Vitalik (2016), although the Ethereum platform operates based on a public blockchain, the permissioned network solution for using the Ethereum platform can be achieved by executing the transactions and contracts only in the limited authorized nodes separate from the Ethereum public blockchain. Although this is a feasible solution for building a permissioned network using the Ethereum platform, this thesis concludes that the infrastructure of the network remains based on the Ethereum blockchain, and thereby the built permissioned network is still using the consensus algorithm of the Ethereum public blockchain. However, as reviewed and discussed in Section 3.1.4, the efficiency of the consensus algorithm designed specifically for the permissioned blockchain is higher than that of the public blockchain. Therefore, in terms of efficiency, for building a permissioned network, although it is feasible to build it in the Ethereum platform, unless the permissioned network cannot operate independently, without using the Ehereum public blockchain (for example, the permissioned network has to use the Ethereum account to do transfers.), then using a specific permissioned blockchain platform is a better choice, such as the Hyperledger Fabric.

Hyperledger Fabric

Hyperledger projects are hosted by Linux Foundation (Cachin, 2016), with the goal to develop and provide open source blockchains and related building tools. IBM and Digital Asset are main contributors to Hyperledger Fabric. Hyperledger Fabric is developed on the infrastructure of permissioned blockchain and provides a modular design for developing permissioned blockchain-based business networks.

Hyperledger Fabric implements its infrastructure with a permissioned blockchain using RAFT as its consensus algorithm. As discussed, RAFT is developed based on Paxos, and thereby is a CFT algorithm, more suitable for private permissioned blockchains with higher level trust nodes. Hyperledger Fabric is planning to incorporate PBFT to replace RAFT as the consensus algorithm for its permissioned blockchain. In a Hyperledger Fabric permissioned blockchain-based network, the nodes can be categorized into two types, ordering and peer nodes. The ordering nodes work similarly to the leader nodes concept in the Paxos and PBFT. All the ordering nodes work together to form an Ordering Service which is responsible for ordering and broadcasting transactions, while the peer nodes are only responsible for verifying and executing the received transactions. The ledger in the Hyperledger Fabric consists of two components. One is called Blockchain Log which is used to log the history of records for all states of all transactions. The other component is called State Database which is used to maintain the current state of the transactions in the network. This allows all the transactions, related operations and their history states, to be traced.

Similar to Ethereum, Hyperledger Fabric provides a modular architecture in the abstract level to simplify the application development, allowing developers to focus on the design of business network logic instead of the implementation of the infrastructure. In the modular architecture of Hyperledger Fabric, the basic object transferred in a network is defined as an Asset. An asset can be used to define anything; as long as an object can be transferred in a business network, it can be defined as an asset. An asset data consists of key value pairs that describe its definition. Therefore, a transaction can be described as a record of state changes of assets. According to Cachin (2016), the implementation of a distributed application in Hyperledger Fabric has two major parts.

One part is the Chaincode which is smart contracts that define assets, and also define and initiate transactions. It is also used for business network participants to interact with the ledger stored in the permissioned blockchain. The business logic of a network is implemented by defining Chaincode by participants. The other part of Hyperledger Fabric is endorsement policy which defines the role of each participant node in the validation process of each network transaction. Intuitively, it defines which nodes are ordering nodes,

which nodes are peer nodes and which nodes do not participate in the validation process of a transaction. In Hyperledger Fabric, the endorsement policy is implemented by Membership Service Provider (MSP) which is a component used to define rules to manage identities of network participants. Participant identities are in the form of a model called Public Key Infrastructure (PKI) which provides secure network communications. The endorsement policy is a concept in the abstract level. MSP and PKI work together to implement the endorsement policy in the network. Another abstract level concept is Access Control List (ACL) which defines the authority each participant has to the access control of assets and transactions in the business network.

In addition, Hyperledger Fabric also provides a solution for privacy protection in the business network. It allows the participants to define a smaller network inside the business network. This smaller network is called a Channel which is composed of multiple participants. The transactions, assets and chaincode in a channel are only transparent to the participants of that channel and a channel is a black box to other nodes in the business network.

3.1.6 Blockchain-Based Event Ticketing System

Some blockchain-based event ticketing systems' research and literature has already been introduced in preceding sections. This section reviews these researches and discusses the solutions presented as well as the problems that still exist or have not been addressed.

The existing solutions for blockchain-based event ticketing systems can be generalized into two categories. One category is based on the permissioned blockchain, while the other is based on the public blockchain.

Tackmann (2017) introduced a secure event ticketing system, based on the permissioned blockchain-based system, where they designed a system to solve the problem of double-selling, fake tickets, invalid ticket reselling and trust of multiple organizers. In their solution, the nodes are held by the event organizers. A permissioned blockchain, provided by Hyperledger Fabric, is used to store, maintain, and process all the transactions including income distribution, resulting in data transparency. In addition, an event's process of

income distribution is ensured since all the transactions executed on the permissioned blockchain are visible and supervised by all the nodes. Lin et al. (2019) introduced a smart contract-based mobile ticketing system with multi-signature and blockchain focused on solving the problem of secure payment and ensuring the authenticity of tickets. In their design, a ticket is designed to be in the form of a QR code which can be easily stored and presented by a mobile application. The ticket's QR code will be digitally signed twice, once by the event host to ensure the authenticity of the ticket, and then again digitally signed by the consumer at the entrance to complete the payment of the ticket and avoid the QR code being stolen. They also use a smart contract that can be issued with a digital signature from the event host and another digital signature from the ticketing agency to store the sales plan information and prove that the sales plan has the agreement of both parties. The ticketing smart contracts designed by Lin et al. (2019) are processed in a permissioned blockchain called EOSIO. In 2020, another permissioned blockchain-based ticketing system was introduced aimed at designing a direct contract system between the event organizers and consumers to prevent agencies from using macros to book massive numbers of tickets (Ko et al., 2020). They implemented their system using a private blockchain of Hyperledger Fabric. In their private blockchain network, only authorized users can join, and all the ticketing transactions are processed inside it to ensure all transactions are supervised to prevent a consumer from booking massive tickets.

Each of these permissioned blockchain-based ticketing system solutions discussed above solved some of the problems in the ticketing system. However, they all have limitations. As reviewed in 3.1.4.2, the data in the permissioned blockchain network is only transparent to its nodes under access control rules. For the participants outside the nodes, the permissioned blockchain is a black box, same as the centralized system. In the above permissioned blockchain-based ticketing system solutions, consumers are the participants outside the nodes. It is impossible to allow every consumer to host a node in the network. Therefore, the transparency of the ticket information to consumers cannot be guaranteed. The data in the network can be modified without the consumer's knowledge. When disputes occur, the rights and the interests of the consumers cannot be ensured. In addition, Tackmann (2017) and Lin et al. (2019) did not mention how to prevent scalping. In the ticket system introduced by Lin et al. (2019), a scalper can forward the QR code to anyone

else to resell the ticket. The system introduced by Ko et al. (2020) only prevents large-scale ticket bookings. But the scalpers can still book tickets in compliance with the rules of the system introduced by Ko et al. (2020) and resell them at higher prices outside the system. Moreover, there are certain risks in the method of completing payment at the entrance in the system introduced by Lin et al. (2019). Malicious attackers can book massive tickets but not complete the payment by not going to the event. This causes much economic loss to the event host.

Isaksson (2018) reviewed the public blockchain-based ticketing system solutions before 2018, which included Aventus, by Aventus Protocal Foundation (2018) and GUTS, by GET Foundation Team (2017). The Aventus team contributed a white paper in 2018 that introduced a platform to provide protocol level ticketing services based on the public blockchain. The GUTS team introduced a similar system in 2017. According to the review from Isaksson (2018), both systems used the public blockchain of Ethereum to store and process ticketing data and transactions to ensure the transparency of the ticketing system. The Aventus and GUTS solutions have many similarities. For privacy protection, they both used the method of asymmetric encryption to generate a key pair and then encrypt the private information with the public key to protect the privacy of the data stored in the public blockchain. For the problem of fake tickets, digital signing technology is used in both Aventus and GUTS solutions. The major difference of these two systems is in their solutions for scalping prevention. The Aventus team believed they could solve this problem by using the face id, credit card or ID card number, as the identity proof of the ticket owner. They also implied that their ticketing system could generate a unique QR code for each ticket at certain times before the beginning of the event to prevent scalping. The solution of GUTS team for this problem was different. GUTS team designed their system to use the consumer's phone number or social media account as identity proof. Isaksson (2018) did a survey on these methods, and the result showed that the majority of the respondents chose the method of using their phone number. Isaksson (2018) also preferred the GUTS team method of using the consumer's phone number. However, Isaksson (2018) argued that the scalping prevention is not a problem that should be solved on chain. For these two systems, Aventus and GUTS, this thesis has different opinions from the point of view of Isaksson (2018). This thesis argues that there are some factors that these two projects have not

considered. First, the method of using complete face id, credit card number or ID card number as proof of ticket owner enables the ticket records to be traced and thereby violates consumer privacy and identity. And for the unique QR code method of scalping prevention introduced by The Aventus Protocol Foundation (2018), the QR code is merely a picture that contains a string of information that can easily be sent to others. For ticket scalping, the scalpers need only forward the received QR code to their buyers. Furthermore, the GET Foundation Team's (2017) method of using the consumer's phone number carries the same risk of traceability and violating the consumers' identities. In addition, another issue of this method is that although the use of consumer's phone number can prevent massive ticket scalping, it cannot completely eliminate it, because a scalper need only forward the verification message received on their phone to the ticket buyers to complete the reselling of a ticket. Therefore, this thesis does not believe that the methods of scalping prevention introduced in the Aventus and GUTS are perfectly valid. Secondly, both of these methods are based on the Ethereum, a public blockchain platform, and in terms of ticketing systems, not all data should be supervised by the public. There are some types of data that should be retained only by the event organizers, such as tickets sales data, and sharing asset of event organizers. This thesis believes that although the privacy of private data can be protected by encrypting it with the data owner's private key, it is not necessary to store and process the non-public data on the public blockchain, because the cost of storing data and processing transactions on the public blockchain is high and the efficiency is low (Section 3.1.4).

Cha et al. (2018) introduced another public blockchain-based event ticketing system solution which focuses on data privacy protection using public blockchain. Cha et al. (2018) used the method of non-interactive zero-knowledge proof (NIZK) introduced by Hao (2017) to protect the consumers from being tracked while the consumer's identity can be verified. In their system, each consumer has a key pair, and the consumer uses its public key to represent its identity. By using NIZK, when the system needs to obtain and verify the identity of a consumer, instead of the consumer directly providing their public key, the consumer needs to generate several parameters that satisfy a specific equation based on their public key, and then provide these parameters to the system. The system uses the provided parameters from the consumer to generate another equation and verify the

consumer's identity by checking if the provided parameters satisfy a designated condition and if the equation is correct. In this method, the parameters that satisfy the conditions generated by the consumer based on the public key are variable, and thus the parameters submitted to the system by the same consumer are different every time. Therefore, the system cannot determine which tickets are purchased by a same consumer. Compared with the method of irreversible hashing, the zero-knowledge proof can not only protect the consumer's original identity data (the consumer public key in this system), but also prevent the consumer's ticket purchasing history from being tracked in the system. In the hashing method, there is only one hash value for the same data. Although the system cannot reverse the original data from the hash value, the system can determine that the transactions with a same hash value have a high probability of being the same consumer (considering the possibility of a hash collision existing, the probability of directing to a same consumer is not 100%). If the method of the zero-knowledge proof is used, the system cannot direct different transactions to a same consumer because the obtained parameters provided by the consumer are different each time. This thesis believes that Cha et al. (2018) solution is effective when using zero-knowledge proof to protect consumers' privacy (ticket purchase history) from being tracked by the system. However, this solution does not cover the problem of scalping prevention. A scalper can directly transfer the public key, representing its identity, to the ticket buyer to complete the reselling.

The above review and discussion show that although each of these blockchain-based event ticketing system solutions solve some of the problems in the event ticketing systems, they all have limitations and problems. Their solutions cannot solve all the research problems illustrated in Chapter 2 at the same time.

3.2 Other Technologies

This section reviews the InterPlanetary File System (Benet 2014), the asymmetric cryptography and the salt technologies, which have been used to build this thesis's hybrid blockchain-based event ticketing system.

3.2.1 InterPlanetary File System

The InterPlanetary File System (IPFS) is a peer-to-peer distributed file system created by Benet (2014) with the goal of connecting to all computer devices. The files stored in the IPFS are content addressed, and thereby the address of a file stored in the IPFS corresponds to the file's content. If the content of the file changes, the address for retrieving the current state of the file changes accordingly, and the file is not able to be retrieved using the old address. Because of this feature, any modification of the file content in IPFS will change the address of the file and invalidate the original address. Therefore, using IPFS to store and distribute files on the public blockchain by storing only the file's IPFS address on the blockchain instead of the complete file, can reduce the size of the data stored on the blockchain, reducing the cost, while not affecting the transparency and traceability of the file. The address of an IPFS file is also called Content Identifier (CID) where the content-addressing is implemented by hashing. The file content is hashed into a hash value using a hash function (default is SHA256), and then the hash value is packaged into a Multihash string. The Multihash string consists of a hash algorithm code (code of SHA256 is 0x12), the length of the hash value (length of a SHA256 value is 32 bytes, thus, code of the length is 0x20) and the hash value. Therefore, the Multihash string of the SHA256 hash value of a file will be in the format of "1220" + 32 bytes SHA256 hash value. Then, the multihash string is encoded with Base58 Encoding Scheme (Nakamoto, 2009) to be the address (CID) of the file. Since the SHA256 is used as the default hashing algorithm in IPFS, the multihash strings of IPFS files start with "1220" by default, which makes the IPFS file address (CID) a 46 character string starting with Qm by default (Qm is the code for 1220 in Base58).

3.2.2 Asymmetric Cryptography

Asymmetric Cryptography, also called public key cryptography, was first implemented by Diffie and Hellman (1976). Although the concept of asymmetric encryption had been introduced previously, no feasible implementation had been introduced. In 1978, Ron Rivest, Adi Shamir and Leonard Adleman invented another implementation of the asymmetric cryptography called RSA, which is the initial letters of their surnames (Rivest

et al., 1978). To this day, RSA is one of the most widely used asymmetric encryption algorithms. Since RSA, a variety of different asymmetric encryption algorithms have been invented. Although the specific implementations of these algorithms are different, the working logic is the same. Asymmetric encryption algorithms use a key pair to encrypt and decrypt data. A key pair consists of a public key and a private key. Data encrypted with the private key can only be decrypted with its corresponding public key. Similarly, data encrypted with the public key can only be decrypted with its corresponding private key.

Asymmetric encryption is also used to generate digital signatures. In 1989, the RSA was used to generate digital signatures by Lotus Notes which was the first marketed digital signature generator. The implementation of digital signatures is to encrypt a piece of data with a private key and use the encrypted ciphertext as the signature of the owner of the private key. To verify a digital signature, the signer's public key will be used to decrypt the digital signature and then compare the decrypted result with the plaintext of the signature. If they match, it proves that the digital signature is real and belongs to the signer.

Most of the existing asymmetric encryption algorithms (including RSA) are vulnerable when they are facing powerful quantum computing attacks. However, there are post-quantum asymmetric encryption algorithms being invented that can withstand quantum computing attacks (Kuznetsov et al. 2017).

3.2.3 Salt

In cryptography, the concept of salt is a random data that is added to the plaintext content before hashing (Morris & Thompson, 1979). According to Morris and Thompson (1979), adding salt to the plaintext to increase the content length before hashing can effectively increase the difficulty of the plaintext being reversed from its hash value. This is because the longer the length of the plaintext content, the more possible the content value corresponding to the same hash value. Salting is widely used in today's systems as a feasible and effective solution for defending pre-computed attack, such as Oechslin (2003) rainbow table attack.

3.3 Summary

This section gives a summary of the literature review as shown in Table 3.2.

Purpose	Conclusion	Reviewed Articles
Introduction of blockchain	Give a general introduction to the origin of blockchain and its basic working principles to help readers understand how this technology is applied in this research.	1. Becker, G. (2008). Merkle signature schemes, merkle trees and their cryptanalysis. 2. Chung et al., (2019). Blockchain network based topic mining process for cognitive manufacturing. Wireless Personal Communications. 3. Dai, W. (1998). B-money. 4. Finney, H. (2004). Rpow-reusable proofs of work. 5. Göbel, J., & Krzesinski, A. E. (2017, November). Increased block size and Bitcoin blockchain dynamics. 6. Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.
Introduction of different types of blockchain	Give an introduction of the division and definition of different types of blockchains and give a comparison based on their features to help readers understand the application of different types of blockchains in this research.	1. Castro, M., & Liskov, B. (1999, February). Practical Byzantine fault tolerance. 2. King, S., & Nadal, S. (2012). Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. 3. Lamport, L. (1983). The weak Byzantine generals problem. 4. Lamport, L. (2001). Paxos made simple. ACM Sigact News, 32(4), 18-25. 5. Larimer, D. (2014). Delegated proof-of-stake (dpos). 6. Leslie, L. (1998). The part-time parliament. ACM Transactions on Computer Systems, 16(2), 133-169. 7. Mingxiao, et al. (2017, October). A review on consensus algorithm of blockchain. 8. Ongaro, D., & Ousterhout, J. In Search of an Understandable Consensus Algorithm. 9. Peck, M. E. (2017). Blockchain world-Do you need a blockchain? This chart will tell you if the technology can solve your problem. 10. Wüst, K., & Gervais, A. (2018, June). Do you need a blockchain?
Introduction of blockchain platforms	Give an introduction of Ethereum which is a public blockchain based platform, and Hyperledger Fabric which is a project providing permisssioned blockchain solutions. Both of them has been used in this research to build an implementation of the proposed solution for conducting experiments.	1. Buterin, V. (2016). Ethereum: Platform Review. Opportunities and Challenges for Private and Consortium Blockchains. 2. Cachin, C. (2016, July). Architecture of the hyperledger blockchain fabric. In Workshop on distributed cryptocurrencies and consensus ledgers (Vol. 310, No. 4). 3. Eastlake, D., & Jones, P. (2001). US secure hash algorithm 1 (SHA1). 4. Tackmann, B. (2017). Secure Event Tickets on a Blockchain. In Data Privacy Management, Cryptocurrencies and Blockchain Technology (pp. 437-444). 5. Vitalik, B. (2013). Ethereum white paper: a next generation smart contract & decentralized application platform.
Review blockchain based event ticketing systems	Review pervious researches on blockchain based event ticketing systems and discuss the advantages and disadvantages of their solutions to demonstrate the necessity of this study.	1. Aventus Protocal Foundation. (2018). A Blockchain-Based Event Ticketing Protocol [White paper]. 2. Cha et al., 2018. A Blockchain-Based Privacy Preserving Ticketing Service. 3. GET Foundation Team. (2017). GUARANTEED ENTRANCE TOKEN Smart Event Ticketing Protocol [White paper]. 4. Hao, F. (2017). Schnorr non-interactive zero-knowledge proof. 5. Isaksson, C., & Elmgren, G. (2018). A ticket to blockchains. 6. Ko et al., 2020. A Design and Implementation of Macro Prevention Ticket Booking System Using Blockchain. 7. Lin et al., 2019. A Smart Contract-Based Mobile Ticketing System with Multi-Signature and Blockchain.
Introduction of other technologies which has been used in this study	Give a review and introduction of InterPlanetary File System (IPFS), asymmetric cryptography and salt technologies, which have been used in the solution to the research problems in this study.	1. Benet, J. (2014). Ipfs-content addressed, versioned, p2p file system. 2. Diffie, W., & Hellman, M. (1976). New directions in cryptography. 3. Kuznetsov et al., 2017. Code-based public-key cryptosystems for the post-quantum period. 4. Morris, R., & Thompson, K. (1979). Password security: A case history. 5. Nakamoto, S. (2009). Base58. 6. Oechslin, P. (2003, August). Making a faster cryptanalytic time-memory trade-off. 7. Rivest et al., 1978. A method for obtaining digital signatures and public-key cryptosystems.

Table 3.2 Summary of the Literature Review

CHAPTER 4: ARCHITECTURAL DESIGN

This chapter presents the overall architecture of the Hybrid Blockchain-Based Event Ticketing System, including an explanation of how the system was designed to solve the research problems previously discussed. A demonstration is presented to show the system's operation including a conceptual model of its structure and time sequence diagrams of its main functions.

4.1 Overall Architecture

This section presents the general architecture of the hybrid blockchain-based event ticketing system as shown in Figure 4.1.

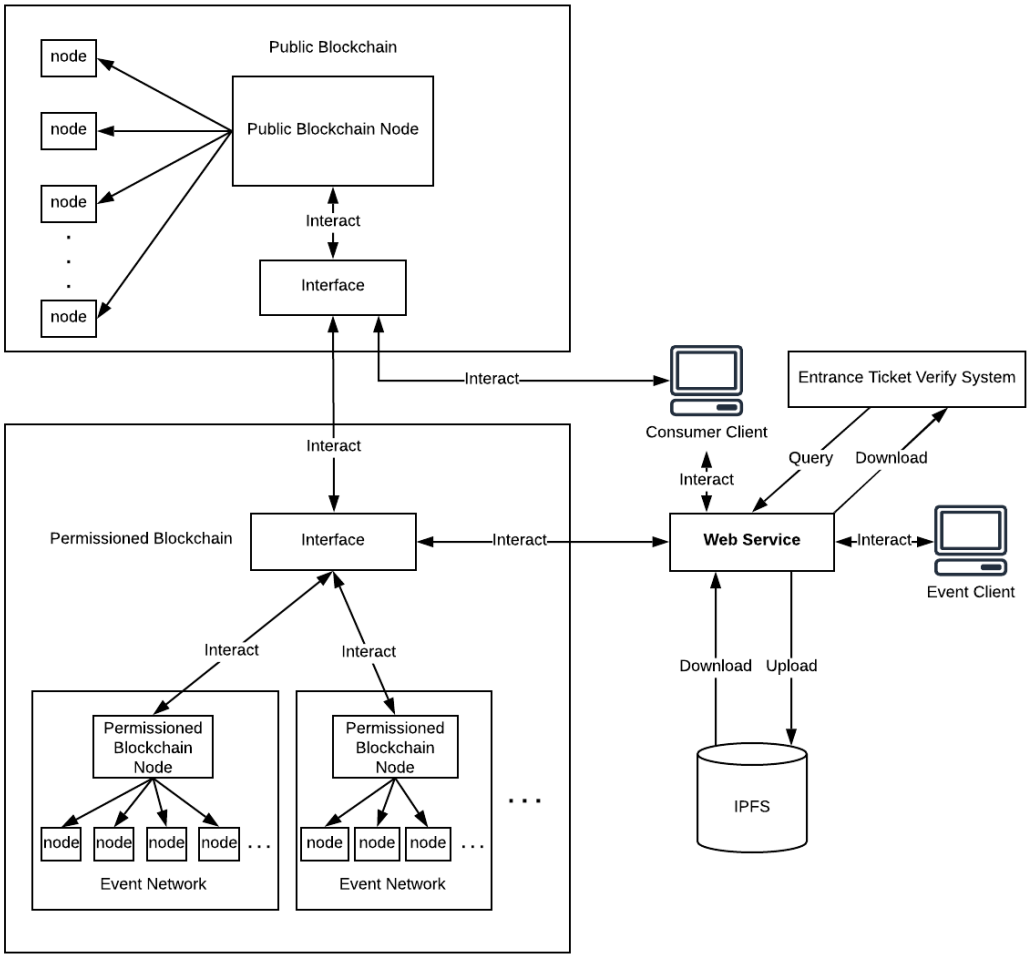


Figure 4.1 General Architecture of the Hybrid Blockchain-Based Event Ticketing System

Figure 4.1 illustrates the general architecture of the event ticketing system comprising seven components, the Consumer Client, the Event Client, the Web Service, the InterPlanetary File System (IPFS) (Benet, 2014), the Entrance Ticket Verify System, the Public Blockchain and the Permissioned Blockchain.

In order to solve the problems of transparency and privacy protection, the architecture was designed to use different types of blockchain to process different types of data to meet the different requirements of transparency and privacy protection. The data which needs to be supervised by the public is processed by the public blockchain component. Protected contracts or evidence data that should not be unilaterally changed is encrypted, and then processed on the public blockchain component. Data that should only be shared among the event participants is processed in its specific event network which is built in the permissioned blockchain component. Considering the high cost of storing data on public blockchain, this thesis used the InterPlanetary File System (IPFS) (Benet, 2014) as a decentralized storage solution, rather than storing the data directly on the public blockchain component, and thereby the public blockchain only stores the address of IPFS files that contain detailed data. The Web Service is designed to work as an interface to the ticketing system, which will be used to process interactions among the ticketing system components. The Web Service will not directly process the protected data. All the protected data will be encrypted into ciphertext before being uploaded to the Web Service. The Web Service is a participant in every event network in the permissioned blockchain. The Consumer Client is a client program with which consumer users can request ticket purchase, while maintaining ticket authenticity verification and servicing ticket refunds. It also securely stores the consumer user's private data such as the consumer user's private key. The Event Client is another client program that provides event participants to register an event and interact with the participated permissioned blockchain-based event networks. In addition, the Event Client can query event data, and keep the participant's private data, such as the participant's event network identity. The Entrance Ticket Verify System is designed to download all the event tickets after ticket sales and check tickets during the entrance of the event audience.

- **Solution to Unforgeable Tickets**

Chapter 2 presented a research goal to build a ticketing system that can solve the problem of fake tickets. One possible method to achieve this goal is to make each ticket traceable, tamper proof after it is generated, and to contain an evidence that cannot be imitated, so as to prove the ticket is authentic. The traceability should allow ticket information to be queried by the consumer after it is sold. This is a necessary condition to ensure that the ticket authenticity can be verified. Similarly, ensuring the ticket information is tamper proof is also another necessary condition in order for the ticket's authenticity to be verified. Of course, if the ticket information can be tampered with, the tamperer can create a fake ticket by modifying the key information on the real ticket, which might not be detected by the ticket verification. Therefore, in order to solve these research problems, the following three questions should be explored.

a. How to make ticket information traceable?

b. How to make the generated ticket information secure?

c. How to design evidence information to verify ticket authenticity that cannot be imitated?

To answer questions a and b, this research uses the blockchain to process and record each ticket transaction. Hence, the blockchain will record all the generated blocks by all the nodes and thereby any transaction written into any block is able to be queried and verified. This application of this feature of blockchain technology to product recording has been discussed in detail in a paper titled "Blockchains as a solution for traceability and transparency" by Jeppsson & Olsson (2017). Tickets can be considered as a special type of product and hence, by using this blockchain feature all the ticket transactions will be recorded on the blockchain and thereby achieve the requirement of ticket record traceability. As reviewed in Chapter 3, question b can also be solved by using blockchain because once a transaction has been written into a block and that block is added into the blockchain, the transaction will be permanently and unalterably recorded. Any modification of the recorded transactions is impossible unless the modifier can pay sufficient price to grow a new branch faster and longer than the

original chain of blocks. For example, in the public blockchain, the price required is half of the sum of computing power of all the nodes.

To answer question c, the evidence should have the ability to prove that the verified ticket is issued under the agreement of the event organizers and this evidence should be unforgeable and unique for each ticket to prove ticket ownership. Digital signature technology can be used to preserve ticket ownership by ensuring a digital signature for each event ticket. The specific steps for generating an event ticket signature are as follows.

Step 1: Each event should generate a public/private key pair that is used to protect its privacy and prove its identity.

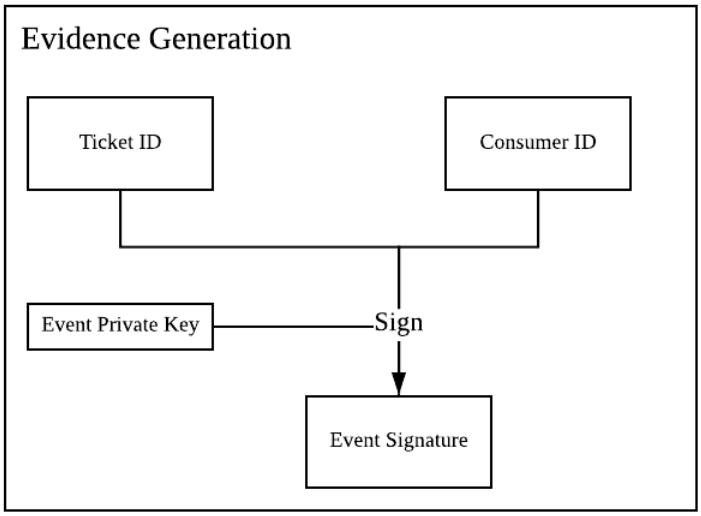
Step 2: The event keeps its private key and provides the public key to the ticketing system.

Step 3: When the system receives a request to generate a new event ticket, a unique ticket ID will be generated for the new ticket.

Step 4: The system should ask the event to sign the ticket by using its private key to encrypt a text containing the issued ticket ID and ticket consumer's ID.

Step 5: The event signature will be recorded as an evidence for the ticket to prove its authenticity.

According to the steps above, an evidence (event signature) of a ticket can be described as in Figure 4.2



Evidence = Encrypt(Ticket ID + Consumer ID, Event Private Key)

Evidence: The evidence used to verify the authenticity of a ticket.

Encrypt(A, K): Encrypt content A with Key K.

Ticket ID: A unique ID for a ticket.

Consumer ID: A unique ID for a consumer.

Figure 4.2 Evidence used to verify the authenticity of a ticket

From Figure 4.2, an evidence contains the ticket ID information and consumer ID which are encrypted by an event's private key. Thus, only the public key of that event can decrypt this evidence. As long as the decrypted information matches the ticket ID and the consumer ID, it proves that the ticket was actually issued by the event for that consumer. In general, this method verifies the authenticity of a ticket simply by using the public key of the event to decrypt the evidence and compare whether the decrypted information matches the ticket information.

- **Solution to Transparency and Privacy Protection**

As discussed in Chapter 2, the centralized system is an opaque black box for the users, which makes it difficult for consumers' rights to be effectively protected. Consumers have

no ability to supervise the system, thus information in the system can be arbitrarily changed by the system holder without the consumers' knowledge. As a result, it is difficult for consumers to obtain evidence to protect their rights and interests. For example, when a dispute arises between the consumer and the system holder regarding the ticket information or the agreements, since the system holder can modify the stored information in the system, it is difficult for the consumer to obtain evidence to protect their interests. Therefore, the consumers' participation is completely dependent on the trust of the system holder. To solve this problem, this research introduces a method that uses the blockchain-based decentralized system to process ticketing. This method is used because unlike centralized systems in which the system holder can manipulate the data as it wishes, in a blockchain-based decentralized system, no data can be changed once it is recorded on the blockchain, nor can it be deleted. In addition, every node has all the blockchain transaction records. Hence, as long as the user chooses to be or access to a node of the blockchain system, the user can query all the transaction records, ensuring the transparency of the system.

The protection of user's privacy is also an important requirement of the ticketing system. Since the use of blockchain technology will make the stored information completely transparent to each node, how to protect user's privacy while ensuring transparency will be discussed. As described in Chapter 2, users in the ticketing system are not just classified as consumers, but event organizers too. Users with different identities have different requirements for information transparency and privacy protection. This research believes that different methods should be designed for different requirements and different types of assets.

In the case of consumers, inevitably, in a ticketing system, some of their private information needs to be stored in the system during the ticketing process. Therefore, consumers should be assured that their private information cannot be read or tampered with by anyone else without their permission. Storing information on a public blockchain can effectively solve the problem of illegally tampering with consumers' data since once the data has been recorded on a public chain, it is impossible to tamper unless the attacker has enough computing power to grow a new branch faster and longer than the original one. Furthermore, an effective access control of the consumers' private information can be

achieved by encrypting the data and then storing the encrypted ciphertext on the public blockchain. This research adopts the asymmetric encryption method to encrypt the consumer's private information whereby consumers will be generated a key pair according to the asymmetric encryption algorithm adopted and their public keys will be used to encrypt their private information. Then, the encrypted ciphertext will be uploaded to the public blockchain where the consensus algorithm will ensure that the data cannot be tampered. Since only the ciphertext is stored on the public blockchain, only the consumer who has the private key is able to decrypt it, thus the consumers' actual private information is protected.

For the event organizers, this research breaks down their information into two categories. The first category is public information about the event, such as the location of the show, the time, venue information, statements and related publicity documents. This public information should be readable and transparent to everyone and should be signed by the event to prove that the information is authentic and undeniably released by the event organizers. A solution to providing public information is to store the information on a public blockchain in plaintext to ensure that the information cannot be tampered, and using the private key of the event to generate a digital signature of the event as an undeniable evidence to prove that this information is actually released by the event.

The second category is the event organizer's private information and shared property which should not be accessible by the public, including the event's private key. The access control of the information in this category should ensure the transparency among all organizers and ensure the data is fully protected from others. This problem too can be achieved by using the public blockchain and encrypting all the private data of the event, while sharing the method of data decryption among all the organizers of the event. However, the requirement is not just storage, it also needs to use this information to do transactions. All these transactions should also be transparent to all participating organizers, hence they should be executed transparently. At the same time, these transactions should be a black box for others. If we choose to use public blockchain to execute these transactions, the content of these transactions should also be encrypted. Although this is possible, when the public blockchain executes these transactions, the content of the transactions must be

decrypted first. Executing encryption and decryption algorithms consumes on-chain computing power and for public blockchains, the cost is high. Considering these requirements, using permissioned blockchain to store the event private data and execute related transactions is a more appropriate solution. In the permissioned chain, only certain groups can become nodes, while in the public chain, everyone can become a node. In terms of event ticketing systems, only the organizers of an event have the right to become a node of the permissioned blockchain-based ticketing network. In this way, the private information of the event and related transactions are ensured to be transparently distributed to all event organizers. Meanwhile, this permissioned blockchain-based ticketing event network is a black box for the outside, which ensures the protection of the privacy of the event. The overall permissioned blockchain-based ticketing network is formed by combining all event networks, which can be directly shown in Figure 4.3.

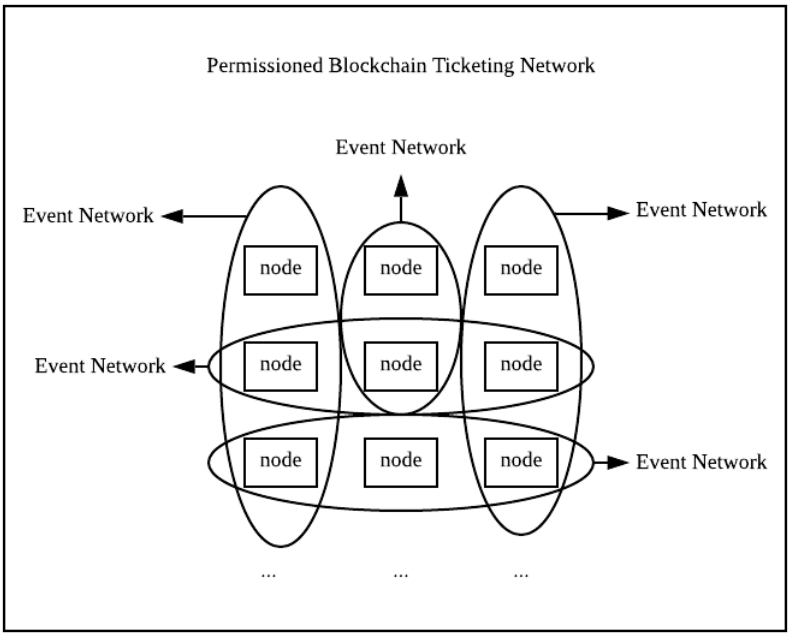


Figure 4.3 Permissioned Blockchain-Based Ticketing Network

In addition to the above, apart from data and transactions belonging to consumers and event organizers, ticket information and its related transactions should also be considered. These ticketing information can be divided into public ticket information used as records of

tickets, and information that should be shared only between the organizers of the event, such as ticket sales. The storing of the former and the execution of the related transactions such as the generation of ticket record data should be performed on the public blockchain. If some of the information relates to the privacy of the user, such as the identity of the ticket purchaser, this information should be encrypted before being executed and stored. For the latter, this information needs to be shared between the organizers and not accessible for the outside. The transparency and privacy requirements of this information are the same as for the privacy information of the event organizers and their shared property which should not be known by the public. Therefore, this information and related transactions should be stored and executed on the permissioned blockchain.

According to the analysis and explanation above, the solution for transparency and privacy protection can be portrayed as follows in Figure 4.4.

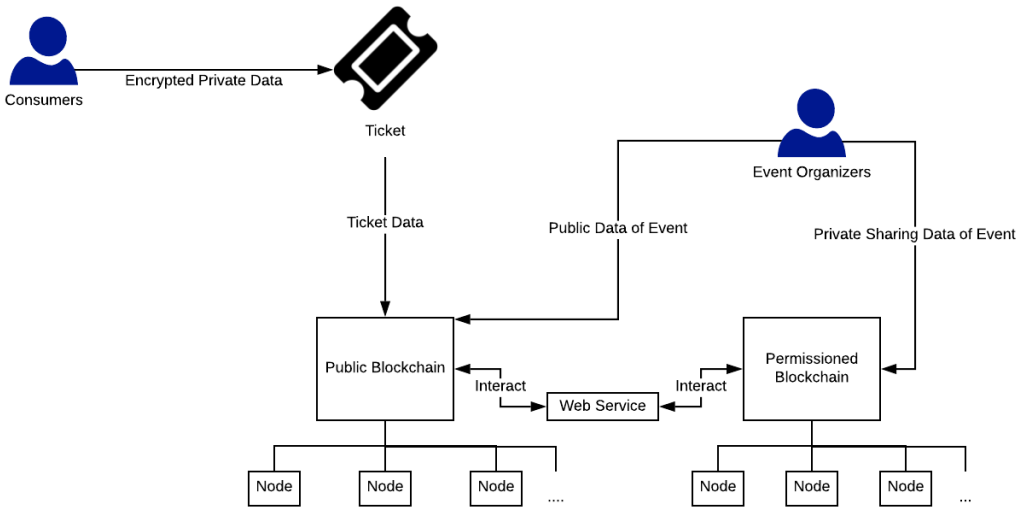


Figure 4.4 Solution for Transparency and Privacy Protection

As illustrated in Figure 4.4, the solution for transparency and privacy protection in the ticketing system can be summarized as using a combination of public blockchain and permissioned blockchain to store different categories of information and execute different types of transactions on different types of blockchain. A public blockchain is used to store the consumers' encrypted private information within the ticket, the public information of the event and execute the related transactions. A permissioned blockchain is used to store

the private information of the event organizers as well as their shared properties and execute the related transactions. Two different type blockchains interact with each other indirectly through the web service component to achieve the consistency of information.

- **Solution to Ticket Scalping Prevention**

Chapter 2 discussed that scalping prevention is an important part in the design of a ticketing system. Illegal ticket reselling results in much economic loss and chaos to consumers, event and ticketing systems. Section 3.1.6 reviewed some of the ticket scalping prevention methods used in other studies. However, these studies overlook some problems in the methodology. These unconsidered problems can be summarized as follows. First, the technical expertise of the ticket scalper should not be underestimated. Previous studies introduced a ticket verification method for generating a QR code for each ticket before the opening of the event, which underestimates the technical ability of the scalper. The QR code is just a picture containing a string of information and thereby QR code images can be easily forwarded to the ticket buyer from the ticket scalper. The scalper can also make a fake client for the buyers of the scalping to present the transferred QR code, because the QR code of each ticket is generated from the information of that ticket which is also known by the scalper as the ticket owner. The same problem also applies to the method of receiving a verification message using a phone. In addition, the scalper can hand over the scalped ticket account purchasing to the ticket buyer since the account itself has no value. Thus, any verification that does not require the ticket owner to be verified can be easily passed by the ticket scalpers to the ticket buyer via the method of transferring account as long as the ticketing account is valueless. This research believes that to prevent scalping, the ticket owner must provide an unforgeable, verifiable and not easily transferable identity information to prevent the scalper from reselling tickets by transferring accounts. In addition to generating a QR code in real time, some other researchers have introduced other methods of using face ID, ID card number and credit card number, as the owner's required identification. Such methods can effectively prevent scalping, but they also create risk of the ticket owner being identified, resulting in a violation of consumer privacy protection.

Following the above analysis, this research explores two effective methods of scalping prevention. The first method is to limit the number of tickets purchased by each consumer user. However, the scalper can still purchase a large number of tickets by creating multiple consumer accounts if the accounts are worthless. In order to prevent this occurrence, each consumer account must correspond to a unique consumer, which requires the consumer to provide information that can prove their identity and cannot be forged or transferred to others. However, his method must also adhere to consumer privacy protection and therefore the same issues are present when using face ID or ID card number or credit card number. The second method is to verify that the attendee is the ticket owner recorded on the ticket at the entrance. Considering consumer privacy protection, this thesis believes that, unlike the first method where the consumer account and the consumer identity must meet the one-to-one correspondence, the owner information recorded in a ticket for entrance verification does not need to correspond to a unique identity, but rather the relationship between them can be one-to-many. Hence, the ticket owner's identity proof recorded on a ticket can correspond to multiple values, so that the ticket owner's identity cannot be reversed from the identity proof. Thus, the complete real world identity of the ticket owner will not be at risk. And although the identity proof information on the ticket corresponds to multiple values, it confines the scalpers to be only able to resell the ticket to the purchasers whose identity information also correspond to the same value, thereby greatly increasing the risk and difficulty of scalping. This is explained in Figure 4.5.

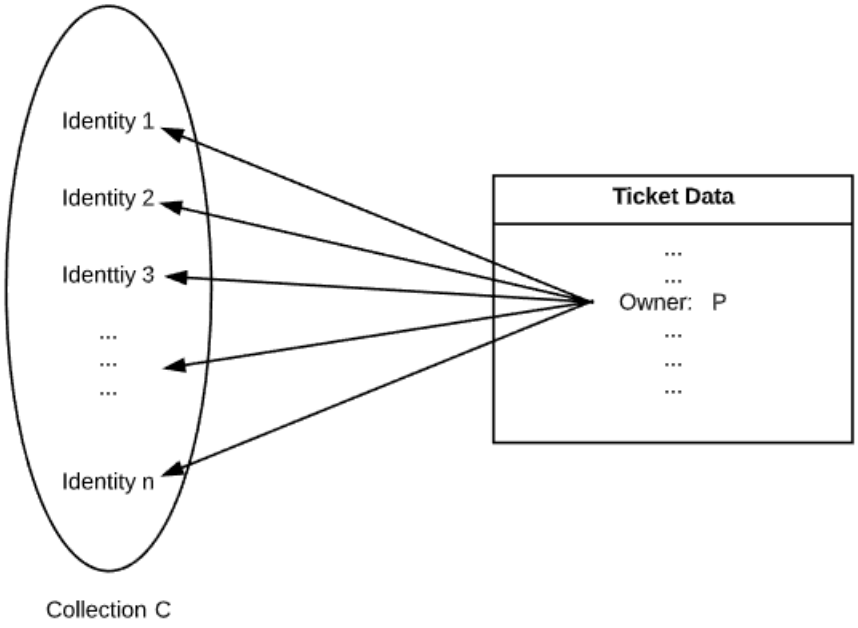


Figure 4.5 Relation between the prove of ticket owner identity and the real identity

In Figure 4.5, P is the value of the identity proof recorded on a ticket. C is a collection of values of possible identities corresponding to the value P. Since all the values in collection C correspond to the value P, even if P is known, it is still impossible to determine which value in collection C is the real world identity of the ticket owner. Thus, the ticket owner's real world identity will not be leaked. At the same time, since only the values in collection C corresponds to P, the ticket scalper can only resell the ticket to a consumer whose identity information is also in collection C. And this will greatly increase the difficulty and risk of scalping tickets since it is uncertain if there is a ticket buyer whose identity information is included in collection C.

The next thing to discuss is how to implement such a relationship in a ticketing system. A common method to achieve a many-to-one relationship is to use a hash function that takes a hash value of ticket owner's identity and records it on the ticket. However, when verifying the identity, considering the efficiency of the entering, it is impossible to manually get the hash value of real world identities and compare it with the record on the ticket. Therefore, the real world identity of each attendee needs to be scanned, hashed, and compared by an offline verification program to avoid being leaked. Nonetheless, for the attendees, this

program cannot be trusted because there is no guarantee that the program holder will not record and leak the scanned real world identity. This research believes that this problem can be solved by scanning only a part of the real world identity instead of the complete number. For example, using an ID card number, the scanning method can be achieved by blocking a part of the barcode on the ID card number to ensure the real world identity of the ticket owner is not leaked. In this method, even if the holder of the verification program records and leaks the scanned information, the complete real world identity of the attendees will still not be leaked, thus, the attendee's real world identity will not be traceable, and the part of the ID card number can still be used to generate the identity proof recorded on the ticket. In order to ensure the transparency of the system, the ticket information is public, which means that everyone has the possibility to access it. In order to protect consumers' privacy, even if it is only a part of the ID card number, its plaintext should not be accessible by everyone. It should only be visible to permissioned participants. Therefore, how to ensure the protection of the ticket owner's partial real world identity information needs to be explored.

For the protection of partial real world identity information, this research gives a solution based on encryption and hashing. In terms of identity verification, it is not necessary to record the plaintext of an identity on a ticket. The ticket need only record a hash value of the partial real world identity. During the verification, the same hash method can be used to hash the scanned partial real identity information, and then compare the result with the identity proof which is the previously generated hash value recorded on the ticket. However, the partial real world identity information simply consists of several digits of numbers, which means it is very easy to be reversed. For example, suppose the last 4 digits of an ID card number are used as the partial real world identity information. Since each digit is a number of 0-9, an attacker who has a hash value of a partial ID card number only needs to try up to 10000 times to find the correct value corresponding to its hash value. To enhance the protection of ticket owner's partial identity information, this research suggests to design a special salting method to salt each identity number before hashing. The salting process can be designed to generate a random string of sufficient length, and then replace some characters of the string with partial identity numbers. Instead of hashing them directly, hashing the salted identity information can effectively reduce the risk of the real identity

numbers being reversed from the hash value. Salting and hashing can be done outside the ticket system, and the consumer only needs to provide the hash value of the salted string to the system. The original salting string and the position of the identity numbers replaced in that salted string can be kept by the consumer until being provided in the entrance verification. The execution of salting and hashing, and the storage of salting information, can be performed by a client program. In this way, except for the consumer, no one else can access the real identity numbers before the entrance ticket verification. Therefore, the partial real identity information is protected.

A life cycle of an identity information can be described as containing the following steps:

1. Consumer provides part of the ticket owner's real world identity number to a client program.

2. The client generates a random string of sufficient length, randomly replacing some characters of the string with the partial identity number. Then the client saves the original random string and the position where the identity numbers were locally replaced.

3. The client program generates a hash value of the salted string, and then sends the hash value to the ticketing system.

4. The ticketing system generates an identity proof of the ticket owner by encrypting the obtained hash value with the target event's public key.

5. The ticketing system issues and records a ticket with the generated identity proof.

6. After ticket sales stop, the event party downloads all the issued ticket information and decrypts all the ticket owner's identity proof with the event private key, and then stores all the decrypted identity proofs, which are the local hash values in Step 3, in the entrance verification system.

7. During the entrance verification, the ticket owner provides its partial identity number, a string that contains the client stored original salting string combined with the position where the identity numbers are inserted.

8. The entrance verification system regenerates the hash value of the salted partial identity number, and then checks if it is included in the decrypted identity proofs saved in step 5.

In order to demonstrate in detail, this thesis gives an example to show the life cycle of an identity information as follows.

For example, a consumer user intends to buy a ticket for a person whose identity number is 12345678.

Step 1: The ticket buyer provides part of the identity number to the client program. 12345678 => 5678

Step 2: The client salts the part of the identity number with a random string of sufficient length.

a) Generate a random string of sufficient length:

Random string: 23f42c67ebd44174a560b0aad18c2f29

(Generate with the method of GUID introduced by Leach et al. (2005))

b) Randomly replace some characters of the random string with the partial identity number which is “5678” in this case as shown in Figure 4.6.

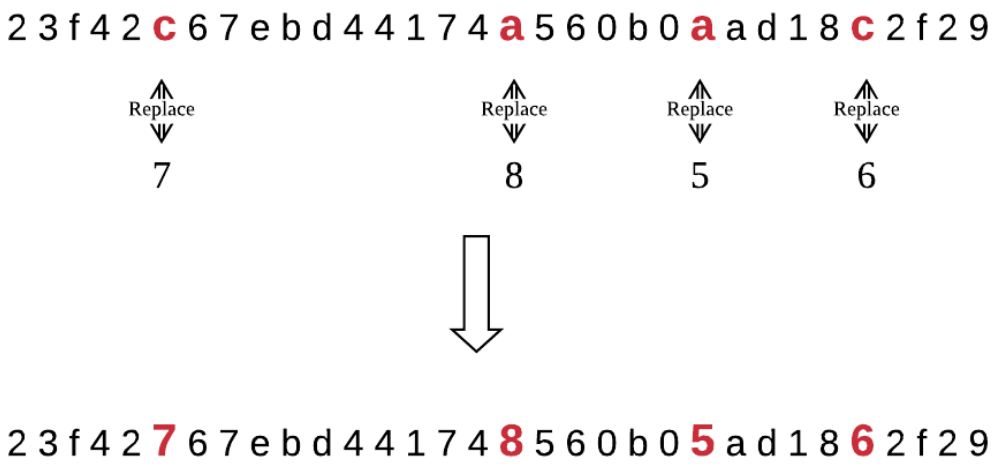


Figure 4.6 Salting Process of Identity Numbers

c) Record the string and the position where the numbers are replaced.

Record the original string: 23f42c67ebd44174a560b0aad18c2f29

Record the replaced position of “5678”: 23-28-6-17

Step 3: The client hashes the salted string in Step 2.

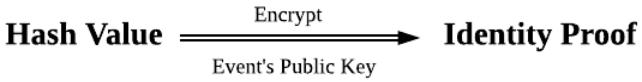
a) The client hashes the salted string:

Hash value = Hash (2 3 f 4 2 7 6 7 e b d 4 4 1 7 4 8 5 6 0 b 0 5 a d 1 8 6 2 f 2 9)

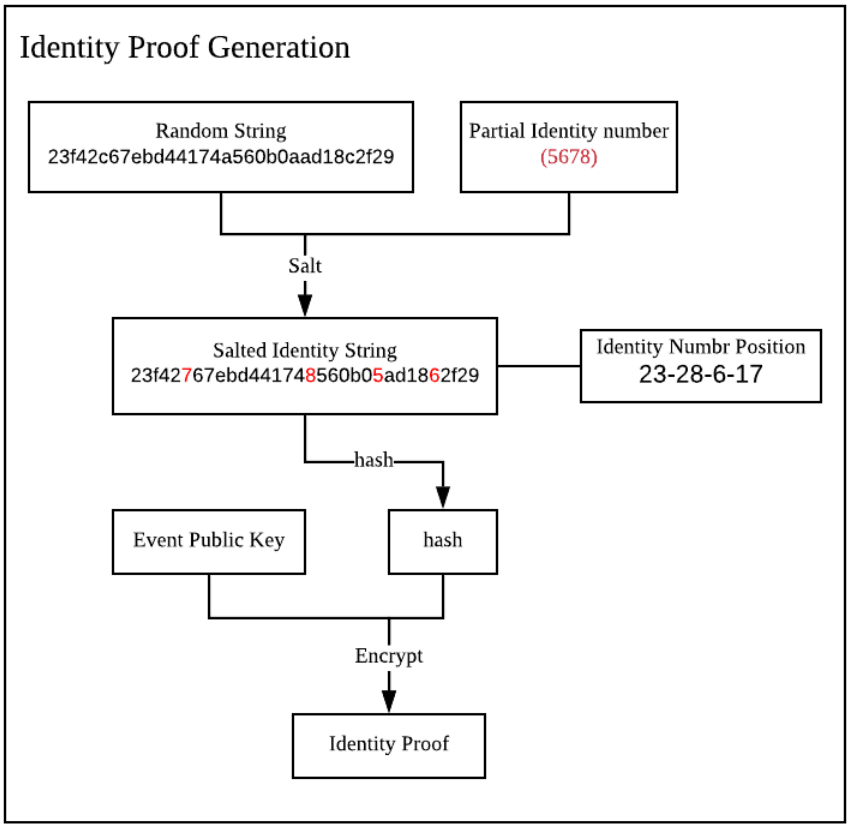
b) Send the hash value to the ticketing system.

Step 4: The ticketing system encrypts the received hash value in Step 3 to generate an identity proof for the ticket owner.

a) The ticketing system encrypts the obtained hash value with the event’s public key and uses the encrypted ciphertext as ticket owner’s identity proof.



The relationship between the Identity Proof and the original partial identity numbers (“5678” in this example) can be described as follows in Figure 4.7.



$$\text{Identity Proof} = \text{Epub}(\text{Hash}(\text{Salt}(5678)))$$

Epub(A): Using event's public key to encrypt A.

Hash(A): Hashing A to a hash value.

Salt(A): Salting A to a salted string.

Figure 4.7 Relationship between Identity Proof and Partial Identity Numbers

Step 5: The ticketing system issues a ticket with the generated identity proof and stores the ticket.

Step 6: After the tickets sales stop.

a) The event party downloads all the tickets from the ticketing system's storage.

b) The event party decrypts all the identity proof on the tickets with the event's private key.

$$\textbf{Identity Proof} \xrightarrow[\text{Event's Private Key}]{\text{Decrypt}} \textbf{Hash Value} = \text{Hash}(\text{Salt}(5678))$$

c) Store the hash values locally in the entrance verification system.

Step 7: During the verification.

a) The entrance verification system scans the ticket owner's partial identity numbers.

b) The ticket owner provides the salting information string that contains the client stored original salting string combined with the position where the identity numbers are replaced. All the information can be integrated into a QR code that represents a string by using separators. For instance, if the separators are "+", the QR code string would be as follows.

QRcode String = Salt String+ID Numbers Position

Applying this example, the QR code string is as follows.

QRcode String = 23f42c67ebd44174a560b0aad18c2f29+23-28-6-17

Step 8: The entrance verification system scans the QR code and regenerates the hash value in Step 3.

a) The entrance verification system separates the scanned QR code string to get the original salting string and the position of the characters of the salting string where identity numbers are replaced.

b) The entrance verification system replaces the characters of the original salting string with the scanned partial identity numbers based on the position information, and then hashes it as follows in Figure 4.8.

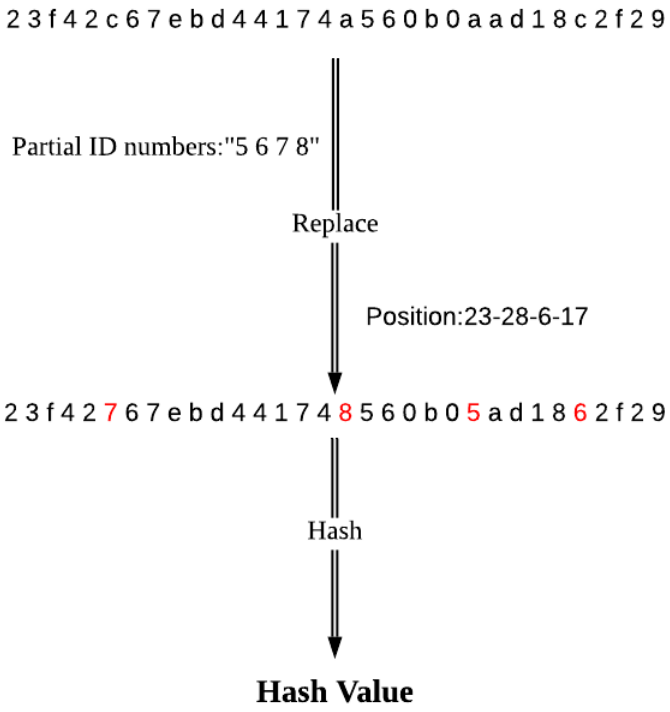


Figure 4.8 Regeneration of the Hash Value of the Salted Identity Numbers

c) The offline verification program checks if the regenerated hash value is included in the storage of Step 6 as shown in Figure 4.9.

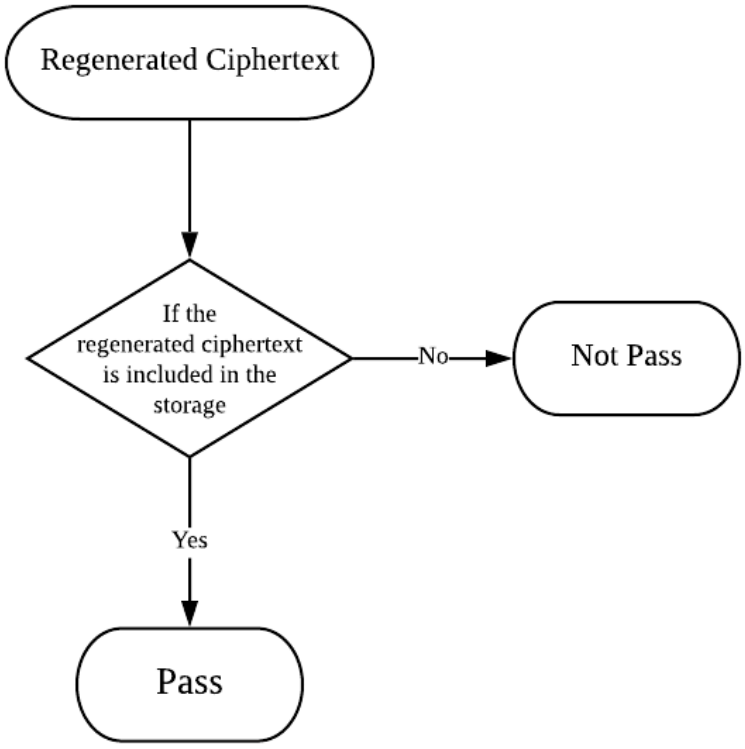


Figure 4.9 Check if the regenerated ciphertext is included in the storage

The Entrance Ticket Verification (Step 6-8) is shown in Figure 4.10.

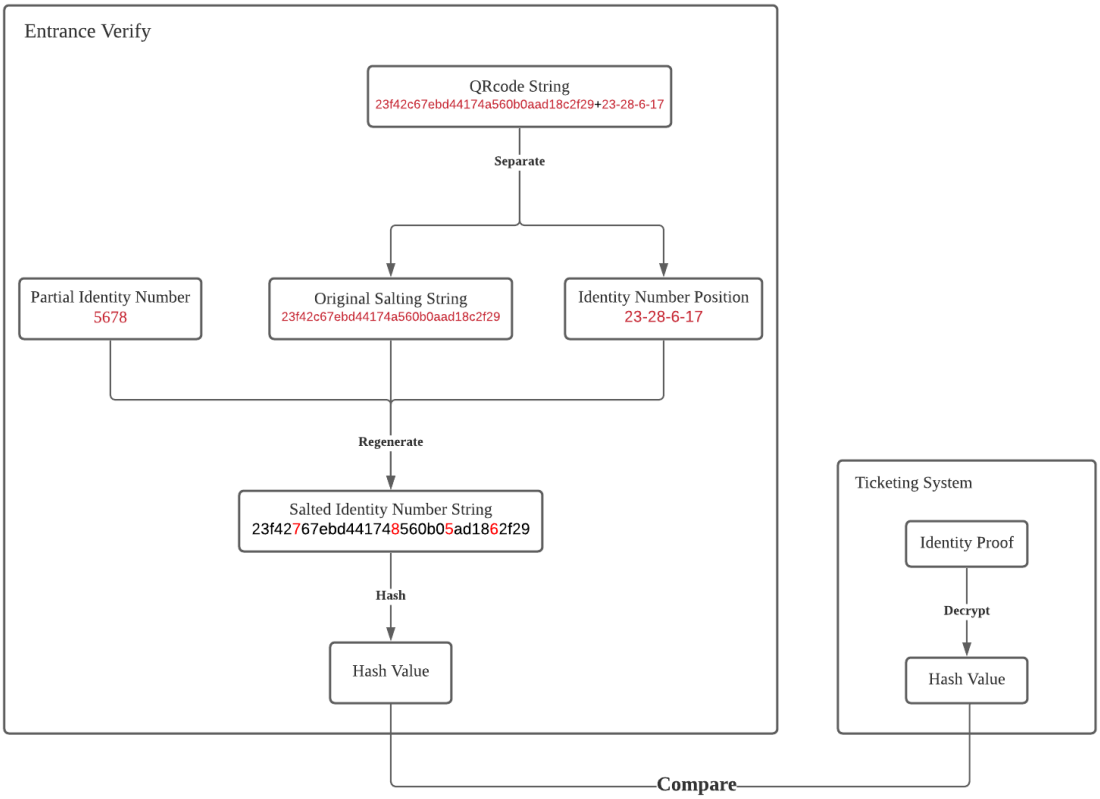


Figure 4.10 Entrance Ticket Verification

4.2 Hybrid Blockchain-Based Event Ticketing System

This section presents a conceptual model of the structure of the Hybrid Blockchain-Based Event Ticketing System, and introduces the working principle of the main functions of the system through time sequence diagrams.

4.2.1 Conceptual Model

The structure of the Hybrid Blockchain-Based Event Ticketing System can be presented by a conceptual model using a class diagram as shown in Figure 4.11.

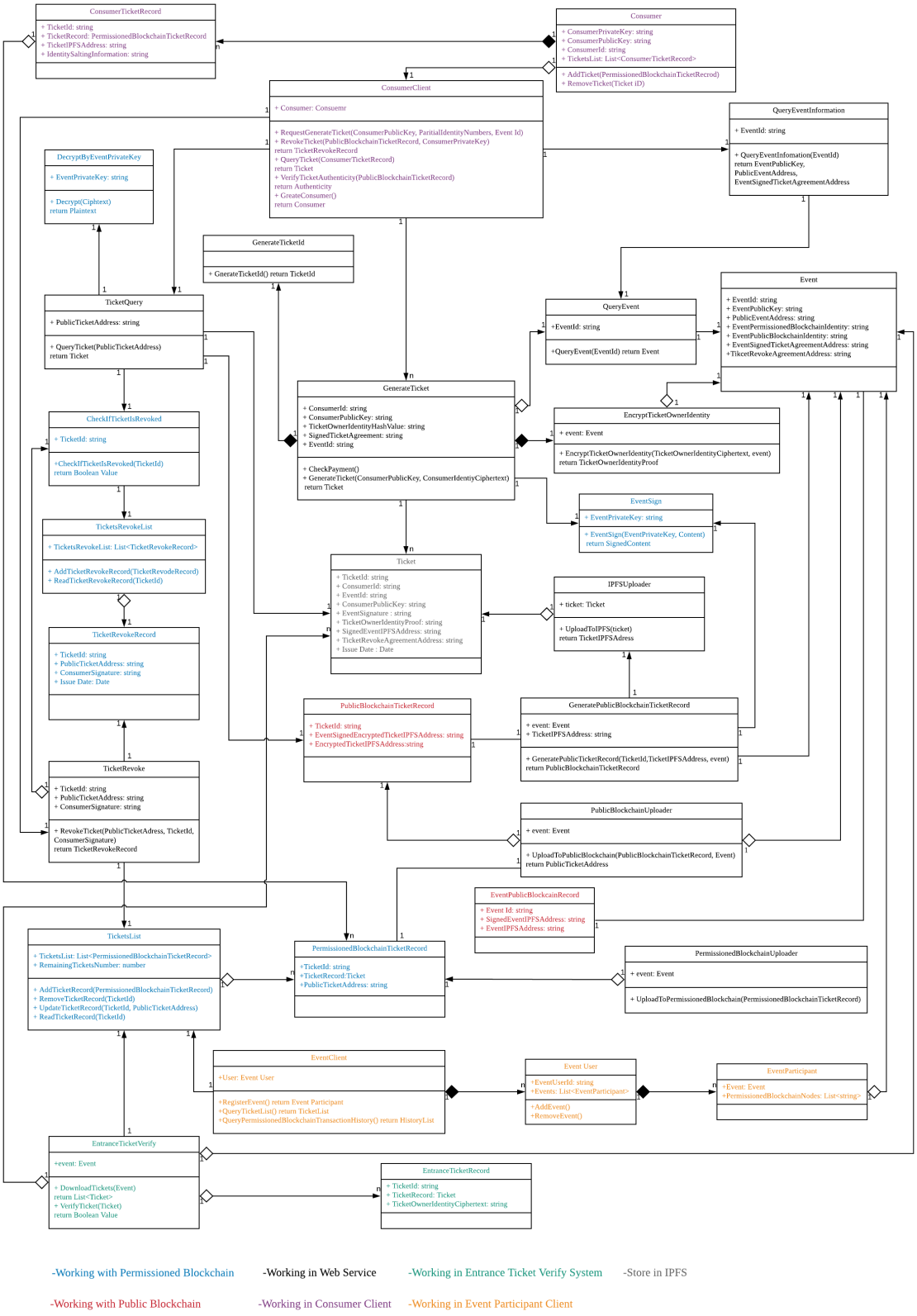


Figure 4.11 Class Diagram of the Hybrid Blockchain-Based Event Ticketing System

As shown in Figure 4.11, the class diagram presents the conceptual structure of the designed system. Each class in the diagram represents a main element working in the system. The color of a class corresponds to the component it works with in the general architecture. Each line represents the relationship between two classes. In the following sections, this thesis describes the role of each class in detail.

Classes in the Consumer Client Component:

Consumer Ticket Record class is a model class regulating the data structure of the ticket information recorded in the Consumer Client component. It has four attributes, which are Ticket ID, Ticket Record, Ticket IPFS Address and Identity Salting Information. The Ticket ID is a unique ID for each ticket. The Ticket Record is a Permissioned Blockchain Ticket Record class object (The Permissioned Blockchain Ticket Record class will be introduced later). The Ticket IPFS Address is an address of an IPFS ticket file containing a Ticket class object (The Ticket class will be introduced later). The Identity Salting Information is a string containing the information of how the ticket owner's partial identity number is salted. It is used to regenerate the ticket owner's identity proof of a ticket.

Consumer class is a model class representing the data structure of a consumer user object. It has attributes such as consumer public key, consumer private key, consumer ID and Tickets List as well as methods of Add Ticket and Remove Ticket. Consumer public key and consumer private key are the key pair used to encrypt, decrypt and digital sign data related to the consumer. The consumer ID is a unique ID for each consumer. The Tickets List is a list of Consumer Ticket Record class objects. The Add Ticket method is used to add a new ticket record into the tickets list attribute. The Remove Ticket method is used to remove an existing ticket record from the tickets list attribute.

Consumer Client class is a controller class implementing major functionalities of the Consumer Client component. It has five methods and a consumer class object saving consumer related data. The function of the Request Generate Ticket method is to send a ticket generation request with the necessary hashed identity data to the Web Service component of the ticketing system. The Revoke Ticket method is used to send a ticket revoking request with a revoke agreement signed by the consumer's private key to the Web Service. The Query Ticket method is used to query a consumer ticket's detailed

information with a Consumer Ticket Record class object. The Verify Ticket Authenticity method is used to initiate a request for consumer to verify the authenticity of a ticket. The function of the Create Consumer method is to generate a new consumer class object for a consumer.

Class in the IPFS component:

The Ticket Class is a model class defining the data structure of the ticket information data saved in IPFS. It has the attributes of Ticket ID, Consumer ID, Event ID, Consumer Public Key, Event Signature, Ticket Owner Identity Proof, Signed Event IPFS Address, Ticket Revoke Agreement Address and Issue Date. The Ticket ID is a unique ID for each ticket. The Consumer ID is the unique ID of the consumer who purchased the ticket. The Event ID is the unique ID of the related event. The Consumer Public Key is the Public Key of the consumer who purchased the ticket, which will also be used to verify consumer's signature. The Event Signature is an evidence to prove the authenticity of the ticket. The Ticket Owner Identity Proof is an encrypted ciphertext which will be used to verify the ticket owner's identity during the entrance check. The Signed Event IPFS Address is an event private key encrypted address of the IPFS file that contains event detailed information uploaded by the event party. The Ticket Revoke Agreement Address is an address of a ticket revocation agreement file of an event stored in the IPFS component. The issue date is a timestamp of the date that the ticket was issued.

Class in the Public Blockchain component:

The Public Blockchain Ticket Record is a model class defining the data structure of the ticket data record by the Public Blockchain component. It has attributes of Ticket ID, Event Signed Encrypted Ticket IPFS Address and Encrypted Ticket IPFS Address. The Ticket ID is the unique ID for each Ticket Class object. The Encrypted Ticket IPFS Address is a ciphertext of an IPFS address of a ticket file encrypted by an event public key. The Event Signed Encrypted Ticket IPFS Address is a digital signature of an event. It is generated by encrypting an Encrypted Ticket IPFS Address with an event private key.

The Event Public Blockchain Record is a model class representing the data structure of event information data recorded by the Public Blockchain component. It is composed of

the attributes of Event ID, Signed Event IPFS Address and Event IPFS Address. The Event ID attribute is a unique ID for each Event Class object. The Event IPFS Address attribute is an address of an IPFS file that contains event basic information data, such as event position, entrance time and event description etc. The Signed Event IPFS Address attribute is a ciphertext of the Event IPFS Address. It works as an event signature to prove the authenticity of the Event IPFS Address.

Classes work in the Web Service component:

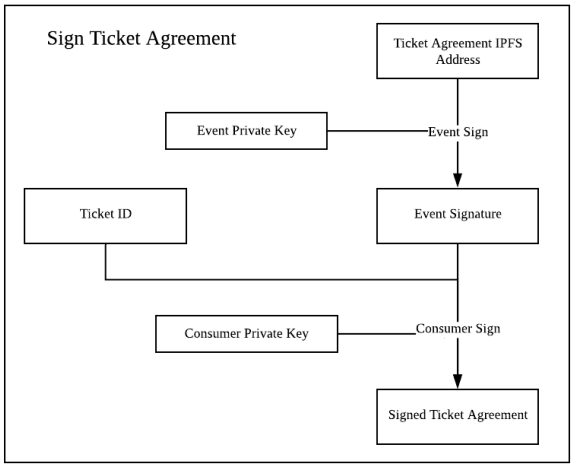
The Event Class is a model class regulating the data structure of the event related data stored in the Web Service component. It has the attributes of Event ID, Event Public Key, Public Event Address, Event Permissioned Blockchain Identity, Event Public Blockchain identity, Event signed Ticket Agreement Address and Ticket Revoke Agreement Address. The Event ID is a unique ID generated for each event. The Event Public Key is the public key of a key pair generated for each event, which will be used to encrypt protected data of the event and verify the authenticity of the event signature by decrypting the data encrypted with the event private key. The Public Event Address is an address of the related Event Public Blockchain Record class object. The Event Permissioned Blockchain Identity is an identity card for the Web Service to interact with the event network in the permissioned blockchain. The Event Public Blockchain Identity is the identity data for the Web Service to upload and query data from the public blockchain. The Event Signed Ticket Agreement Address is an event private key encrypted address of the event ticketing agreement file stored in the IPFS component. The Ticket Revoke Agreement Address is an address of a ticket revoke agreement file for an event stored in IPFS component.

The Query Event Class is a controller class containing an attribute of Event ID and a method of Query Event. The Event ID is the unique ID of each Event related to the Event Class. The Query Event method is used to query a stored Event Class object by Event ID. It returns an Event Class Object.

The Query Event Information is a controller class with an attribute of Event ID and a method of Query Event Information. The Event ID is the same as the attribute under the same name in Event Class. The Query Event Information is different with the Query Event method of the Query Event Class. Instead of returning a complete Event Class object, the

Query Event Information method only returns the attributes of Event Public Key, Public Event Address and Event Signed Ticket Agreement Address of an Event Class object.

The Generate Ticket Class is a controller class in the Web Service component to generate the Ticket Class data to be stored in the IPFS component with the necessary data uploaded from the Consumer Client component. It has the attributes of Consumer ID, Consumer Public Key, Ticket Owner Identity Hash Value, Signed Ticket Agreement and Event ID. The Consumer ID and Consumer Public Key are from the consumer who requests to generate the ticket. They are uploaded from the Consumer Client Component. The Ticket Owner Identity Hash Value is a hash value of a salted string related to the potential ticket owner's identity. The Signed Ticket Agreement is an encrypted IPFS address of the ticketing agreement file with a Ticket ID. It is digitally signed by the consumer private key and the event private key. The generation method of the Signed Ticket Agreement data is to use the consumer private key to encrypt the Event Signed Ticket Agreement Address data of the Event Class and the Ticket ID as shown in Figure 4.12. The Event ID is the unique ID of the event to which the ticket belongs.



SignedTicketAgreement= Cpri(Epri(IPFS Address of the Agreement File)+Ticket ID)

Epri(A): Using event's private key to encrypt A.

Cpri(A): Using Consumer's private key to encrypt A.

Figure 4.12 Generation of a Signed Ticket Agreement

The Generate Ticket Class also has two methods, which are Check Payment Method and Generate Ticket Method. The function of Check Payment method is to check if the payment of the ticket is completed correctly. Since this research focuses on solving the research problems defined in Chapter 2, it will not give a detailed implementation for the payment processing and checking. The Generate Ticket method is used to generate the necessary data with consumer inputs and stored Event Class data to generate a new Ticket Class object to be stored in the IPFS. It returns a Ticket Class object.

The Generate Ticket ID Class is a controller class which has a method of Generate Ticket ID. The Generate Ticket ID method is used by the Generate Ticket method of the Generate Ticket Class to generate a unique ID as the ticket ID for a Ticket Class object. It returns a unique ticket ID.

The Encrypt Ticket Owner Identity Class is a controller class with an attribute of event and a method of Encrypt Ticket Owner Identity. The event attribute is an Event Class object to provide an event public key. The Encrypt Ticket Owner Identity method is used to generate a Ticket Owner Identity Proof of a Ticket Class object by encrypting a Ticket Owner Identity Ciphertext of a Generate Ticket Class object with an event public key.

The IPFS Uploader Class is a controller class with an attribute of ticket and a method of Upload To IPFS. The attribute ticket is a Ticket Class object. The function of the Upload To IPFS method is for the Web Service component to upload a Ticket Class object to IPFS as a file. It returns an IPFS address of the uploaded ticket file.

The Generate Public Blockchain Ticket Record Class is a controller class containing attributes of event and Ticket IPFS Address, as well as a method of the Generate Public Ticket Record Class. The attribute event is an Event Class object. The Ticket IPFS Address is an IPFS address of a ticket file returned by the Upload To IPFS method of an IPFS Uploader Class object. The Generate Public Ticket Record method is used to generate a Public Blockchain Ticket Record Class object to be recorded in the Public Blockchain component. It returns a Public Blockchain Ticket Record Class object.

The Public Blockchain Uploader Class is a controller class containing an attribute of event and a method of Upload To Public Blockchain. The attribute of event is an event object

used to provide event public blockchain identity for the Web Service to interact with the Public Blockchain component. The Upload To Public Blockchain method is used to initiate a smart contract on the Public Blockchain component to record a Public Blockchain Ticket Record Class object. It returns an address of a public blockchain transaction which records a Public Blockchain Ticket Record Class object.

The Permissioned Blockchain Uploader is a controller class with an attribute of event and a method of Upload To Permissioned Blockchain. The attribute of event is an event object to provide event permissioned blockchain identity data for the Web Service component to interact with the event network in the Permissioned Blockchain component. The Upload To Permissioned Blockchain is used to upload a ticket record containing an address of a public blockchain ticket record which is a return value of the Upload To Public Blockchain method of a Public Blockchain Uploader Class object.

The Ticket Query Class is a controller class composed of an attribute of Public Ticket Address and a method of Query Ticket. The attribute of Public Ticket Address is an address of a public blockchain ticket record. The function of the method of Query Ticket is to query a Ticket Class object with a Public Ticket Address. It will first interact with the Public Blockchain component to access the public blockchain ticket record. Then, it will interact with the Permissioned Blockchain component to check if the ticket is revoked and decrypt the encrypted Ticket IPFS Address with the event private key. Then, it will download the ticket file with the decrypted Ticket IPFS Address from the IPFS component and return a Ticket Class object.

The Ticket Revoke Class is a controller class containing attributes of Ticket ID, Public Ticket Address, Consumer Signature and a method of Revoke Ticket. The Ticket ID is a unique ID for each Ticket Class object. The Public Ticket Address is an address of a public blockchain ticket record. The Consumer Signature is a ciphertext of an address of a ticket revoke agreement file for an event stored in the IPFS component and a Ticket ID encrypted with a consumer private key. It is used as an undeniable evidence for the revocation of a ticket to prove that the consumer user agrees to revoke the ticket. The Ticket Revoke method is used to interact with the Permissioned Blockchain component to generate a

revoke record for a ticket. It returns a Ticket Revoke Record Class object. The Ticket Revoke Record Class will be introduced later in this thesis.

Classes working with the Permissioned Blockchain component:

The Permission Blockchain Ticket Record Class is a model class representing the data structure of the ticket related data records in the Permissioned Blockchain component. It has attributes of Ticket ID, Ticket Record and Public Ticket Address. The Ticket ID is a unique ID for each Ticket Class object. The Ticket Record is a Ticket Class object. The Public Ticket Address is an address of a public blockchain ticket record.

The Ticket Revoke Record Class is a model class describing a data structure used to record a revocation of a ticket in the Permissioned Blockchain. It is composed of attributes of Ticket ID, Public Ticket Address, Consumer Signature and Issue Date. The attributes of Ticket ID, Public Ticket Address and Consumer Signature are the same as the attributes under the same name of the Ticket Revoke Class. The Issue Date attribute is a timestamp of the generation of a Ticket Revoke Record Class object.

The Tickets Revoke List Class is a model class containing an attribute of Ticket Revoke List as well as Add Ticket Revoke Record method and Read Ticket Revoke Record method. The attribute of Ticket Revoke List is a list of Ticket Revoke Record Class objects. It is used to store all the records of ticket revocation. The Add Ticket Revoke Record method is used to add a Ticket Revoke Record Class object into the list. The Read Ticket Revoke Record method is used to read a Ticket Revoke Record Class object in the list.

The Check If Ticket Is Revoked Class is a controller class which has an attribute of Ticket ID and a method of Check If Ticket Is Revoked. The Ticket ID attribute is a unique ID for each Ticket Class object. The Check If Ticket Is Revoked method is used to check if a ticket has already been revoked with a ticket ID. It will query a Ticket Revoke List Class object to check if the designated ticket ID is included in the ticket revoke list. It returns a Boolean value representing the result.

The Tickets List Class is a model class composed of attributes of Tickets List, Remaining Tickets Number as well as Add Ticket Record method, Remove Ticket Record method, Update Ticket Record method and Read Ticket Record method. The attribute of Tickets

List is a list used to store all the valid tickets records of an event. The attribute of Remaining Tickets Number is a number that counts how many tickets are left for an event. The Add Ticket Record method is used to add a Permissioned Blockchain Ticket Record Class object into the tickets list and add one to the Remaining Tickets Number. The Remove Ticket Record method is used to remove a Permissioned Blockchain Ticket Record from the tickets list by a ticket ID and decrease the Remaining Tickets Number by one. The Update Ticket Record method is used to update a Permissioned Blockchain Ticket Record Class object in the tickets list with a new Public Ticket Address attribute's value. The Read Ticket Record method is used to read a Permissioned Blockchain Ticket Record Class object of the tickets list.

The Event Sign Class is a controller class with an attribute of Event Private Key and a method of Event Sign. The Event Private Key attribute is a private key of a key pair for an event. The function of the Event Sign method is to generate a signature of an event by encrypting designated content to a ciphertext with the Event Private Key.

The Decrypt By Event Private Key Class is a controller class containing an attribute of Event Private Key and a method of Decrypt. The Event Private Key is the same as the attribute under the same name of the Event Sign Class. The Decrypt method is used to decrypt a ciphertext encrypted by the event public key with the Event Private Key.

Classes working in Entrance Ticket Verify System component:

The Entrance Ticket Verify System component also contains an Event Class which is a model class the same as the Event Class contained in the Web Service component.

The Entrance Ticket Record class is a model class regulating the data structure of the ticket record stored in the Entrance Ticket Verify System component. It includes three attributes, which are Ticket ID, Ticket Record and Ticket Owner Identity Ciphertext. The Ticket ID is the unique ID for each ticket. The Ticket Record is a Ticket class object containing ticket detailed information. The Ticket Owner Identity Ciphertext is the event public key decrypted Ticket Owner Identity Proof of the Ticket Record attribute, which will be used in the entrance ticket verification.

The Entrance Ticket Verify Class is a controller class implementing the major function of the Entrance Ticket Verify System component. It is composed of an attribute of event as well as methods of Download Tickets and Verify Ticket. The event attribute is an Event Class object. The function of Download Tickets method is to download all the valid ticket records of an event. It will first interact with the Permissioned Block to get a tickets list containing all the valid ticket records of an event network. Then, for each ticket record included in the obtained tickets list, it will use the Query Ticket method of the Ticket Query Class to query the ticket stored in the IPFS component with the Public Ticket Address attribute of each Permissioned Blockchain Ticket Record. The Download Tickets method returns a list of Ticket Class objects. The function of Verify Ticket is to regenerate a ticket owner's identity proof to check if it is included in the downloaded tickets list. The Verify Ticket method returns a Boolean value as the result.

Class in the Event Client component:

The Event Participant Class is a model class regulating the data structure of a record for an event identity in the Event Client component. It has attributes of Event and Permissioned Blockchain Nodes. The event is an Event class object. The Permissioned Blockchain Nodes is a list of addresses for nodes in an event network in the Permissioned Blockchain component.

The Event User Class is a model class describing the data structure of a record of a user in the Event Client component. It has attributes of Event User ID and Events as well as methods of Add Event and Remove Event. The Event User ID attribute is a unique ID for each user in the Event Client component. The Events attribute is a list of Event Participant class objects used to record the identities of different event networks owned by a user in the Permissioned Blockchain component. The Add Event method is used to add an Event Participant class object into the Events attribute. The Remove Event method is used to remove an existing Event Participant class object from the Events attribute.

The Event Client Class implements the major functionalities of the Event Client component. It has an attribute of User as well as methods of Register Event, Query Ticket List and Query Permissioned Blockchain Transaction History. The User attribute is an Event User class object used to represent a user in the Event Client component. The Register Event

method is used to allow a user request to register an event. The Query Ticket List method is used to query the list of all the ticket records of an event from the event network of the Permissioned Blockchain component. It returns a Ticket List Class object. The Query Permissioned Blockchain Transaction History is used to query all the transactions' history records of an event network of the Permissioned Blockchain component. It returns a list of transactions' history records.

From the discussion above, it is apparent that a ticket for an event is recorded in different components. This thesis explains how the designed system ensures the consistency of the ticket records of a same ticket in different components. The record of an event ticket in the designed system can be summarized as shown in Figure 4.13.

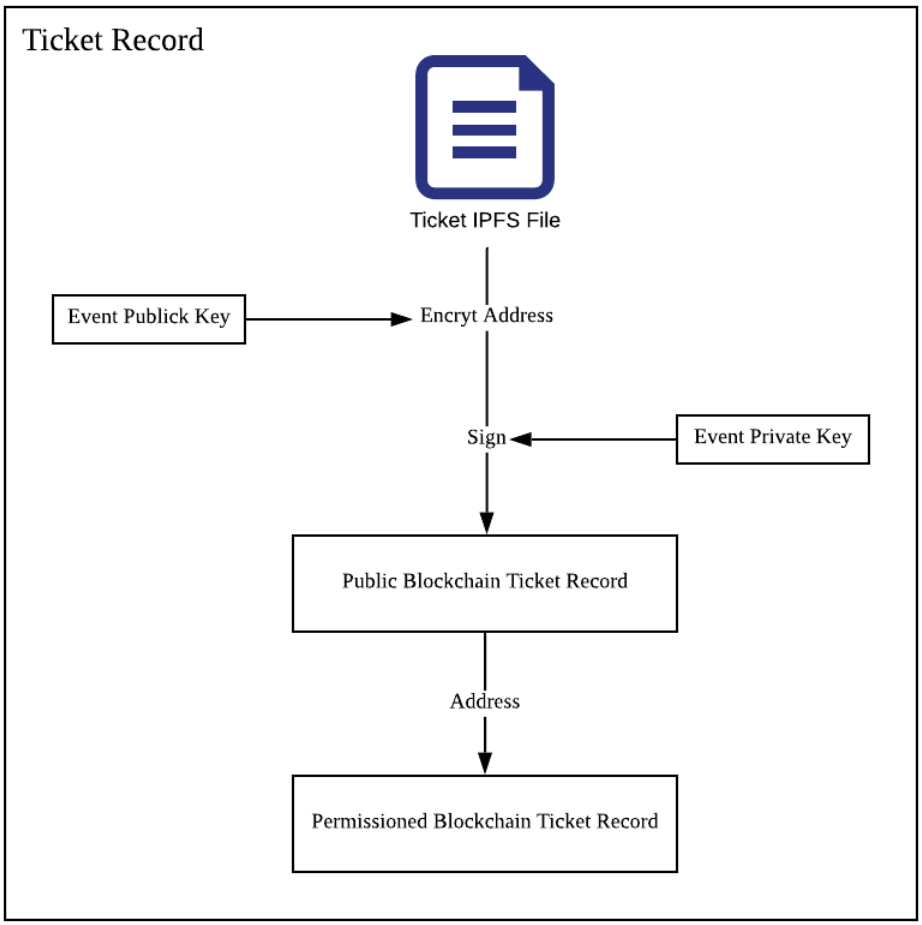


Figure 4.13 Ticket Record in Hybrid Blockchain-Based Event Ticketing System

As shown in Figure 4.13, a ticket is recorded in three records in the designed system, which are Ticket IPFS file, Public Blockchain Ticket Record and Permissioned Blockchain Ticket Record. The detailed information of a ticket is stored in an IPFS file. The address of the IPFS file is encrypted and then stored in a public blockchain transaction along with an event signature signed for the encrypted IPFS ticket file address to relate the public blockchain transaction to the IPFS ticket file. The address of the public blockchain transaction is stored in a permissioned blockchain transaction to relate it to the public blockchain ticket record. These three ticket records are linked in this way to form a complete ticket record in the designed system, which ensures the consistency of the ticket records of a same ticket.

The above has described in detail the function of each class in the conceptual model. In the following, this thesis explains how the solutions to research problems discussed in Section 4.1 are implemented in the designed architecture and conceptual model.

For privacy protection and transparency, all the privacy data of an event, such as ticket sales, records of ticket revocation, are stored in the event network of the Permissioned Blockchain component which is a black box for others who do not have an identity card of that event network. The privacy data of an event can only be accessed by a participant who has an identity card of that event. In this way, the privacy of an event is protected. Meanwhile, the Permissioned Blockchain component ensures that the data stored in an event network and all the transactions using those data can be recorded by every node in that event network. Every participant with an identity card of that network can access all the records and history. This ensures the transparency among participants of an event. In terms of the public information, such as the basic information of an event, the designed architecture stores them in an IPFS file, and then records the file address on the Public Blockchain component to ensure the transparency and immutability of the public information. For the privacy data of a consumer, as can be seen in the conceptual model, the plaintext of a ticket owner's partial identity number will be only stored in the Consumer Client component which is fully controlled by the consumer user as a client program. All the other components will only store an encrypted identity proof ciphertext which cannot be reversed. For each ticket, similar to public information data, the ticket information data

is also stored in an IPFS file. However, instead of directly recording the file address, the address of the ticket IPFS file will be encrypted with the event public key before recording it on the Public Blockchain component to prevent the detailed ticket information from being tracked by others. The encrypted IPFS address ciphertext is recorded with an event signature in the Public Blockchain component. In this way, the public blockchain ticket record is supervised by all the nodes and cannot be tampered, which ensures the transparency and traceability for the consumer user. And the consumer's ticket purchase record is protected as the consumer's privacy by encrypting the address of the ticket file containing detailed information. The event signature for the encrypted address ensures the authenticity of the ticket.

4.2.2 Ticket Generation Process

This section presents the process of generating a ticket in the designed system based on the general architecture and the conceptual model using a time sequence diagram as shown in Figure 4.14.

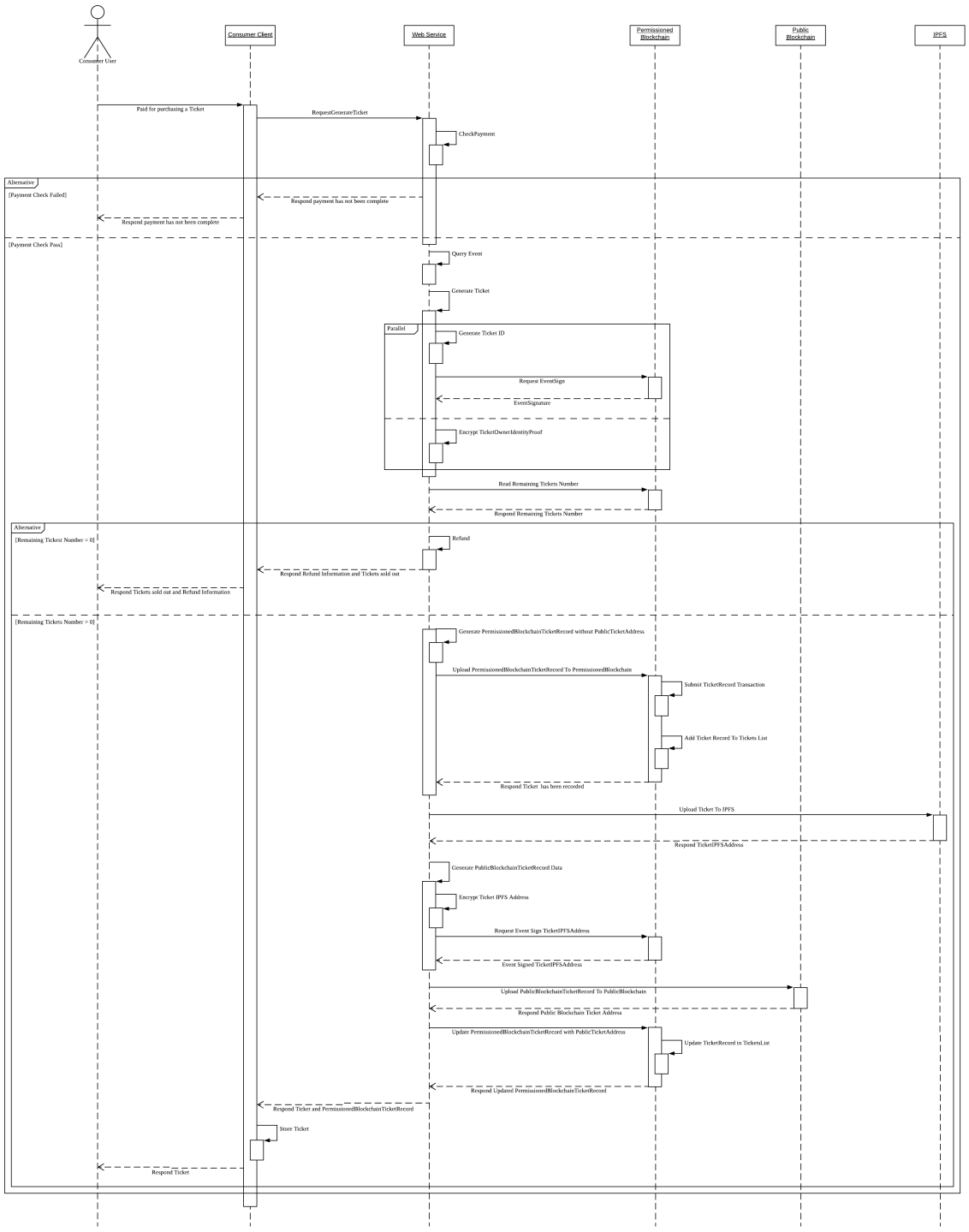


Figure 4.14 Time Sequence Diagram of the Ticket Generation Process

As can be seen in Figure 4.14, there are five components and an actor participant in the ticket generation process, including the Consumer Client component, the Web Service component, the Permissioned Blockchain component, the Public Blockchain component,

the IPFS component and the Consumer User as an actor. The interactions between these components to generate a ticket in the designed system can be described by the following steps.

Step 1: The Consumer Client component sends a request with a generated ticket owner's identity hash value to the Web Service component to request to generate a new ticket after a Consumer User paid for purchasing a ticket.

Step 2: After receiving the request, the Web Service Component calls the Check Payment method (in the Generate Ticket class) to check whether the payment is complete. If so, move on to execute the following steps. If not, the Web Service component returns a message to inform the Consumer Client component that the payment is not completed. Then the Consumer Client responds to the Consumer User the same message.

Step 3: If the payment is completed, the Web Service Component calls Query Event method (in the Query Event class) to get the related Event Class object with detailed information corresponding to the given event ID and do the next steps.

Step 4: The Web Service component calls the Generate Ticket method (in the Generate Ticket) to generate a Ticket Class object.

a) Call the Generate Ticket ID method (in the Generate Ticket ID class) to generate a unique ID and access to the related event network in the Permissioned Blockchain component to call the Event Sign method (in the Event Sign class) to generate an event signature, while encrypting the obtained ticket owner identity ciphertext with the event public key at the same time.

b) Use the above data to generate a Ticket Class object.

Step 5: The Web Service component accesses the related event network in the Permissioned Blockchain component to read the value of the Remaining Tickets Number attribute of the Tickets List Class object of the event.

Step 6: If the Remaining Tickets Number in Step 5 equals 0, the Web Service Component refunds the ticket (this thesis only focuses on ticketing process, and will not discuss the implementation of refunding), and then responds to the Consumer Client component with

the refund information and a message that all the tickets have been sold out. After this, the Consumer Client responds the same information and message to the Consumer User. If the remaining tickets number is greater than 0, the Web Service component will do the next steps.

Step 7: The Web Service component generates a Permissioned Blockchain Ticket Record Class object whose Public Ticket Address attribute is “waiting”, and then submits it to the event network in the Permissioned Blockchain component.

a) Generate a Permissioned Blockchain Ticket Record class object with the generated ticket ID, the generated Ticket Class object in Step 4 and a string “waiting” as temporary Public Ticket Address attribute’s value.

b) Call the Upload To Permissioned Blockchain method (in the Permissioned Blockchain Uploader Class) to submit a transaction on the event network in the Permissioned Blockchain component to record the generated Permissioned Blockchain Ticket Record class object.

c) Call the Add Ticket Record method (in the Ticket List Class) to add the submitted Permissioned Blockchain Ticket Record Class object into the Tickets List Class object of the event network in the Permissioned Blockchain component.

Step 8: The Web Service component calls the Upload To IPFS method (in the IPFS Uploader class) to upload a file with the generated Ticket class object data to the IPFS component, and then the method returns an address of the IPFS ticket file.

Step 9: The Web Service component calls the Generate Public Ticket Record method (in the Generate Public Blockchain Ticket Record) to generate a Public Blockchain Ticket Record class object with the returned IPFS ticket file address in Step 9.

a) Encrypt the returned address of the IPFS ticket file in Step 9 with the event public key.

b) Send a request to the related event network in the Permissioned Blockchain component to call the Event Sign method (in the Event Sign Class) to generate an event signature for the encrypted address.

c) Use the above data to generate a Public Blockchain Ticket Record class object.

Step 10: The Web Service component calls the Upload To Public Blockchain method (in the Public Blockchain Uploader class) to submit a smart contract to record the generated Public Blockchain Ticket Record class object on the Public Blockchain component, and then returns the address of the submitted smart contract.

Step 11: The Web Service component updates the Public Ticket Address attribute's value of the related ticket record in the event network in the Permissioned Blockchain component with the returned address of the ticket record on the Public Blockchain component in Step 10.

a) The Web Service component accesses the related event network in the Permissioned Blockchain component and calls the Update Ticket Record method (in the Tickets List Class) to update the Public Ticket Address attribute's value of the related Permissioned Blockchain Ticket Record Class object in the Ticket List Class object of the event with the returned address of the ticket record in the Public Blockchain component in Step 10.

b) The Update Ticket Record method (in the Tickets List Class) returns the updated Permissioned Blockchain Ticket Record Class object to the Web Service component. And then the Web Service component responds to the generated Ticket Class object in Step 5 as well as the returned Permissioned Blockchain Ticket Record Class object to the Consumer Client component.

Step 12: The Consumer Client component calls the Add Ticket method (in the Consumer class) to generate a Consumer Ticket Record Class object with the returned ticket records in Step 12 and add it into the tickets list of the corresponding Consumer class object. Then the Consumer Client component responds with the Ticket class object to the Consumer User.

In the designed ticket generation process, the ticket record on the permissioned blockchain is generated in Step 7 without a valid value of the Public Ticket Address attribute. After the ticket record on the public blockchain is generated in step 10, the ticket record on the permissioned blockchain is updated with a valid address of the ticket record on the public blockchain in Step 11. This is to shorten the time that a ticket generation thread occupies the Remaining Tickets Number which is a resource that can only be occupied and updated

by one thread at the same time to ensure the consistency. In this way, when a ticket generation thread completes step 7, the occupied resource (Remaining Tickets Number in Tickets List Class) can be released, and then it can be used by another thread immediately instead of waiting for the ticket record on the public blockchain to be generated, uploaded and recorded, which is a time-consuming process.

It can be seen from the above steps that in the ticket generation process, the consumer's private data can only be directly accessed by the Consumer Client component. In the other components, it is processed and stored in the form of encrypted hash value ciphertext which is irreversible. The storage of the event private data and all the calls to this data are executed in the event network on the Permissioned Blockchain component. All the operations of the data in the permissioned blockchain will be recorded by all the nodes of the event network, which ensures traceability and transparency of the event private data and operations among the event participants. Meanwhile, the permissioned blockchain is a black box to the others, which protects the privacy of the event private data. The generated ticket (a Ticket class object) is stored as a file in the decentralized IPFS component. Its IPFS addresses are encrypted and stored on the Public Blockchain component to ensure that it is immutable, transparent and supervised by its consumer. During the ticket generation process, the Web Service is the only centralized system in the Hybrid Blockchain-Based Event Ticketing System. However, the Web Service component does not store or process private data in plaintext form.

4.2.3 Ticket Authenticity Verification Process

This section presents a time sequence diagram to illustrate the process of consumer users to verify the authenticity of their tickets as shown in Figure 4.15.

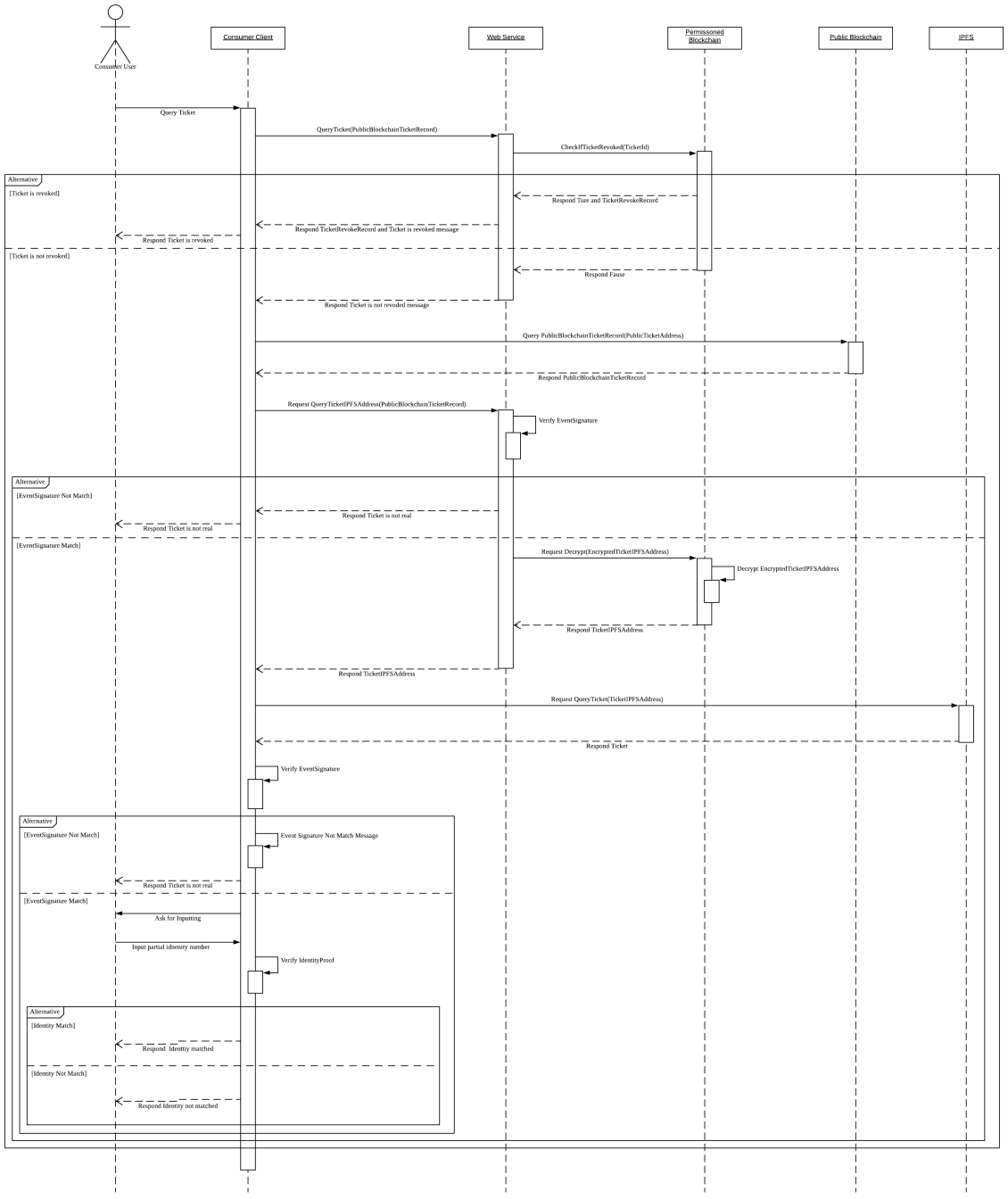


Figure 4.15 Time Sequence Diagram of the Ticket Authenticity Verification Process

As can be seen in Figure 4.15, there are five components and an actor involved in the ticket authenticity verification process, including the Consumer Client component, the Web Service component, the Permissioned Blockchain component, the Public Blockchain

component, the IPFS component and the Consumer user as an actor. The process of ticket authenticity verification can be described as including the following steps.

Step 1: When a Consumer user intends to verify the authenticity of a ticket, the Consumer Client component calls the Verify Ticket Authenticity method (in the Consumer Client class) to start the ticket authenticity verification process. Inside the method, the Consumer Client component calls Query Ticket method (in the Consumer Client class) to send a request to the Web Service component with a ticket ID and an Event ID.

Step 2: The Web Service receives the request, and then calls the Check If Ticket Is Revoked method (in the Check If Ticket Is Revoked class) to check if the target ticket has already been revoked

a) Call the Query Event method (in the Query Event class) to query the related Event class object to get the detailed information of the target Event with the Event ID.

b) Use the Event Permissioned Blockchain Identity to access the related event network and call the Check If Ticket Is Revoked method with the ticket ID on the Permissioned Blockchain component.

c) If the ticket is revoked, return true and the corresponding Ticket Revoke Record class object to the Web Service component, and then the Web Service component returns the same data to the Consumer Client component, the Consumer Client component responds to the Consumer User that the ticket is revoked. If the ticket is not revoked, return false to the Web Service component, then the Web Service component returns false to the Consumer Client component. The Consumer Client goes ahead to do the next steps.

Step 3: The Consumer Client component sends a request to the Public Blockchain component with an address (the address is contained in the Ticket Record attribute of the related Consumer Ticket Record class object) to query the smart contract containing the target ticket record to get a Public Blockchain Ticket Record class object.

Step 4: The Consumer Client component verifies the event signature of the obtained Public Blockchain Ticket Record class object in Step 3.

a) Send a request to the Web Service component to call the Query Event Information method (in the Query Event Information class) to get the event public key.

b) Use the event public key to decrypt the event signature and compare the decrypted content with Encrypted Ticket IPFS Address attribute of the obtained Public Blockchain Ticket Record class object. If they are not matched, respond to the Consumer User that the ticket is not real. If they are matched, do the next steps.

Step 5:

a) The Consumer Client component sends a request to the Web Service component to decrypt the Encrypted Ticket IPFS Address attribute of the obtained Public Blockchain Ticket Record class object.

b) The Web Service component accesses the related event network and calls the Decrypt method (in the Decrypt By Event Private Key class) on the Permissioned Blockchain component to decrypt the address of the IPFS ticket file.

c) The Permissioned Blockchain component returns the decrypted address to the Web Service component, then the Web Service component returns the address to the Consumer Client component.

Step 6: The Consumer Client component downloads the ticket file containing a Ticket class object from the IPFS component with the returned IPFS ticket address in Step 5.

Step 7: The Consumer Client component verifies the event signature of the obtained Ticket class object in Step 6.

a) Decrypt the event signature with the event public key.

b) Compare the decrypted content with the ticket ID and the consumer ID of the Ticket class object. If they are not matched, respond to the Consumer User that the ticket is not real. If they are matched, do the next step.

Step 8: The Consumer Client component verifies if the ticket identity proof of the Ticket class object is matched with the ticket owner’s partial identity number.

a) The Consumer Client component asks the Consumer User to input the ticket owner's partial identity number.

b) The Consumer Client component regenerates the ticket owner's identity proof with the input partial identity number, identity salting information of the related Consumer Ticket Record class object and the event public key.

c) Compare the regenerated ticket owner's identity proof with the Ticket Owner Identity proof attribute of the Ticket class object. If they are not matched, respond to the Consumer User that the ticket owner's identity is not matched. If they are matched, respond that the ticket is real and the ticket owner's identity is matched.

It can be seen that during the authenticity verification process, the designed system verifies the event signature twice, once for the Public Blockchain Ticket Record class object, and once for the Ticket class object stored in the IPFS ticket file. The reason for this is to prevent a malicious scalper from using the real event signature of a Ticket class object with a mismatched Ticket Owner's Identity Proof to forge an IPFS ticket file. The event signature of the Public Blockchain Ticket Record class object can ensure that the Ticket IPFS Address on it is real. The event signature of the Ticket class object can prove that this ticket is issued by the event for this specific consumer so that when there is a dispute about this ticket, the consumer can use it to prove that this ticket is indeed issued for them.

4.2.4 Entrance Ticket Verification Process

This section presents the process of the implementation of the entrance ticket verification of an event in the designed system using a time sequence diagram as shown in Figure 4.16.

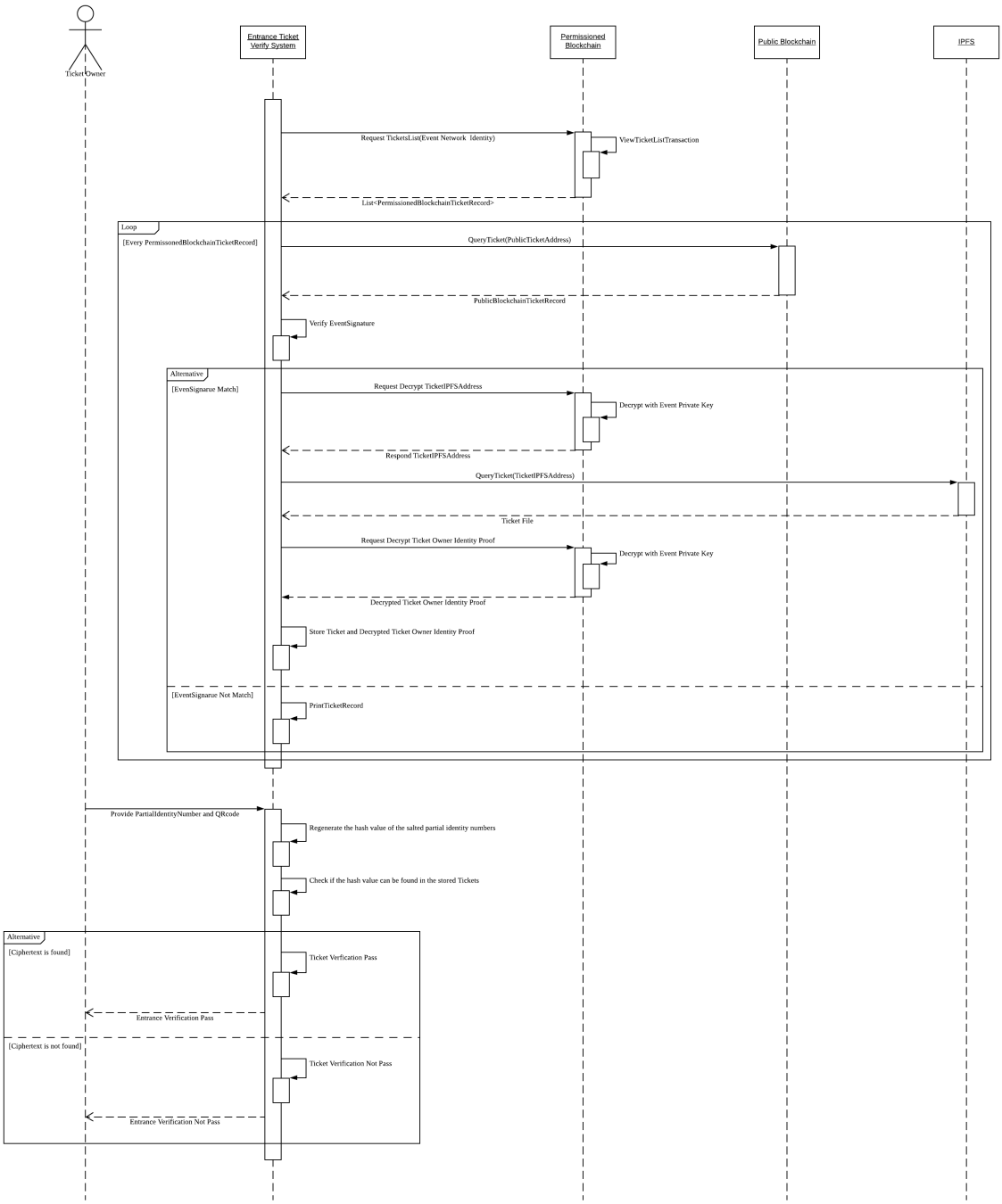


Figure 4.16 Time Sequence Diagram of the Entrance Ticket Verification Process

As can be seen in Figure 4.16, there are four components and an actor involved in the entrance ticket verification process, including the Entrance Ticket Verify System component, the Permissioned Blockchain component, the Public Blockchain component, the IPFS component and the Ticket owner as an actor. The process can be divided into two

sub-processes, which are downloading tickets process and entrance ticket checking process. The downloading tickets process is that after the completion of ticket sales, the Entrance Ticket Verify System component downloads all the valid tickets of the event from other components and stores them locally. The entrance ticket checking process is the process of checking the ticket of each ticket owner before entering the event. The detailed steps are described as follows.

Downloading Tickets Process:

Step 1: The Entrance Ticket Verify System component calls the Download method (in the Entrance Ticket Verify class) to start the downloading tickets process. The Entrance Ticket Verify System component uses an event network participant identity to access the event network and read the Tickets List class object of the event on the Permissioned Blockchain component.

For each Permissioned Blockchain Ticket Record class object included in the returned Tickets List class object, do Step 2 to Step 6.

Step 2: The Entrance Ticket Verify System component uses the Public Ticket Address of the Permissioned Blockchain Ticket Record class object to query the smart contract containing the warrant ticket record on the Public Blockchain component. It will return a Public Blockchain Ticket Record class object.

Step 3: The Entrance Ticket Verify System component verifies the event signature of the returned Public Blockchain Ticket Record class object.

a) Decrypt the event signature with the event public key.

b) Compare the decrypted content with the Encrypted Ticket IPFS Address of the returned Public Blockchain Ticket Record class object. If they are not matched, print the ticket ID. If they are matched, do the next steps.

Step 4: The Entrance Ticket Verify System component accesses the event network and calls the Decrypt method (in the Decrypt By Event Private Key class) to decrypt the Encrypted Ticket IPFS Address of the returned Public Blockchain Ticket Record class object in Step 2. It will return the decrypted ticket IPFS address.

Step 5: The Entrance Ticket Verify System component queries the IPFS ticket file containing a Ticket class object with the returned ticket IPFS address on the IPFS component.

Step 6: The Entrance Ticket Verify System component accesses the event network and calls the Decrypt method (in the Decrypt By Event Private Key class) to decrypt the Ticket Owner Identity Proof of the Ticket class object on the Permissioned Blockchain component.

Step 7: The Entrance Ticket Verify System component generates an Entrance Ticket Record class object with the Ticket class object as Ticket Record attribute and the decrypted Ticket Owner Identity Proof as Ticket Owner Identity Ciphertext attribute, then stores the generated Entrance Ticket Record class object locally.

Entrance ticket checking process:

Step 1: The Entrance Ticket Verify System component scans a ticket owner's partial identity number and a QR code string containing the salting information of a ticket identity proof.

Step 2: Then Entrance Ticket Verify System component calls the Verify Ticket method (in the Entrance Ticket Verify class) to start the entrance ticket checking process. The Entrance Ticket Verify System component salts the scanned partial identity number with the scanned QR code string containing the salting information, and then hashes the salted string to regenerate the Ticket Owner Identity Hash Value.

Step 3: The Entrance Ticket Verify System component checks if the regenerated Ticket Owner Identity Hash Value is included in the ticket records stored in the local storage of the Entrance Ticket Verify System. If it is included, respond a pass message to let the ticket owner enter. If it is not, respond a not pass message to prohibit the ticket owner's entrance.

The downloading tickets process occurs after the tickets sales are completed, which is earlier than the entrance ticket checking process, and which can shorten the time for ticket checking. In the entrance ticket checking process, the Entrance Ticket Verify System component only interacts with the ticket owner, which does not need the participation of

other components in the designed system. Thus, the entrance ticket checking process can be performed offline.

4.2.5 Ticket Revocation Process

This section explains the process of revoking an existing ticket in the designed system using a time sequence diagram as shown in Figure 4.17. The presented process only focuses on the ticket revocation itself and will not involve the refund process.

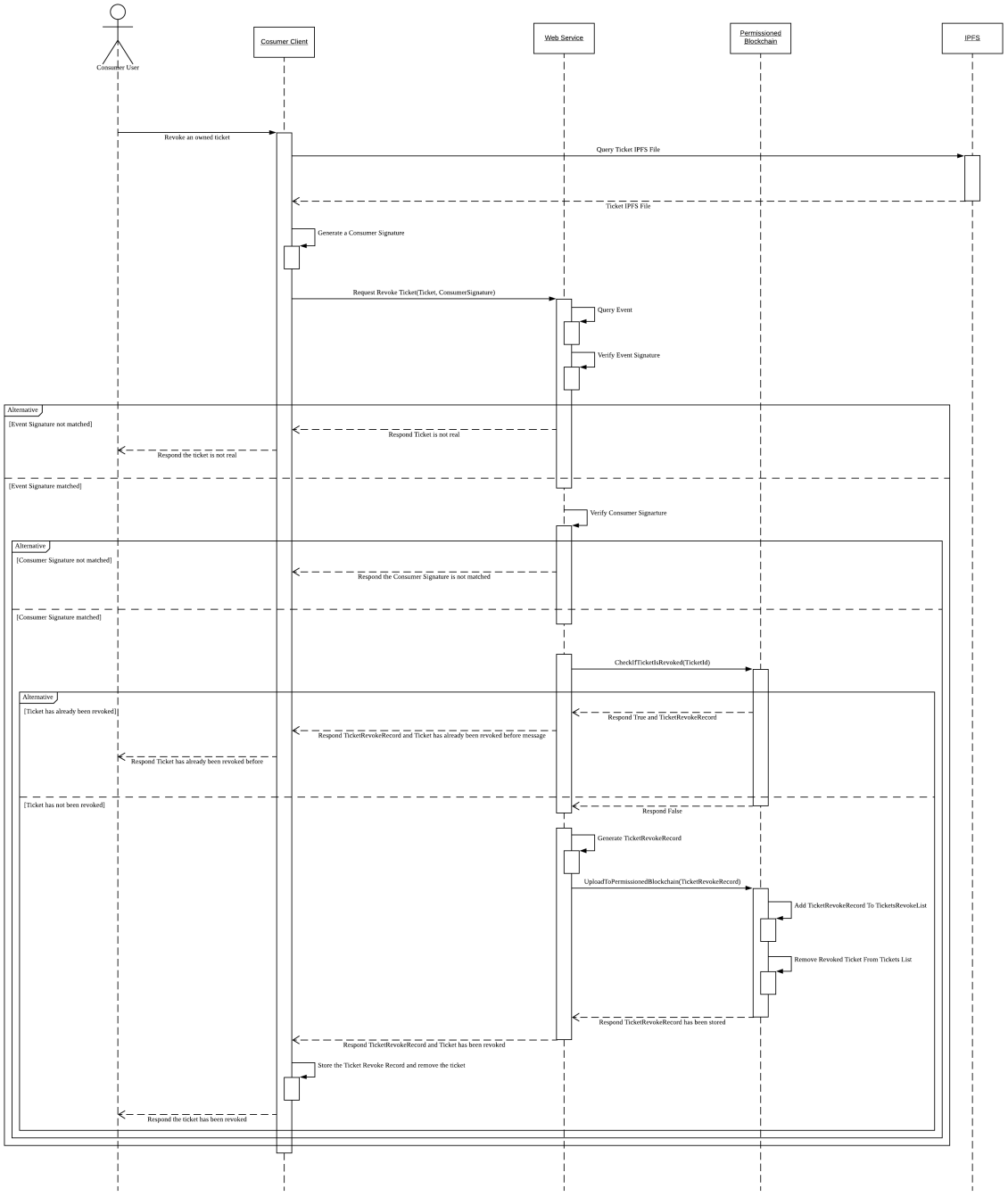


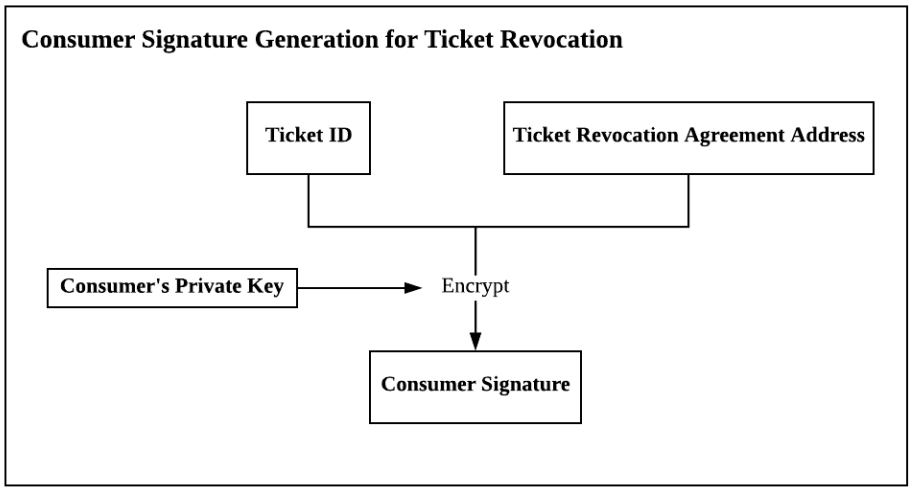
Figure 4.17 Time Sequence Diagram of the Ticket Revocation Process

As can be seen in Figure 4.17, there are four components and an actor participating in the ticket revocation process, including the Consumer Client component, the Web Service component, the Permissioned Blockchain component, the IPFS component and the

Consumer User as an actor. The steps in the ticket revocation process can be described as follows.

Step 1: The Consumer User intends to revoke a purchased ticket. The Consumer Client component calls the Revoke Ticket method (in the Consumer Client class) to start the ticket revocation process. The Consumer Client component downloads the IPFS ticket file containing a Ticket class object from the IPFS component with the Ticket IPFS Address of the related Consumer Ticket Record class object. It will return a Ticket class object.

Step 2: The Consumer Client component generates a consumer signature for ticket revocation by using the consumer private key of the related Consumer class object encrypting the Ticket Revoke Agreement Address of the returned Ticket class object in Step 1 and the Ticket ID as shown directly in Figure 4.18.



Consumer Signature= Cpri(TicketRevokeAgreementAddress+Ticket ID)

Cpri(A): Using Consumer's private key to encrypt A.

Figure 4.18 Consumer Signature for Ticket Revocation

The Consumer Signature for Ticket Revocation can be used as an evidence to prove that the revocation of the target ticket is approved by the Consumer User.

Step 3: The Consumer Client component sends a request to the Web Service component to revoke the target ticket with the returned Ticket class object in Step 1, the generated

consumer signature in Step 2 and the Public Ticket Address of the ticket record attribute (a Permissioned Blockchain Ticket Record class object) of the related Consumer Ticket Record class object.

Step 4: The Web Service component verifies the Event Signature of the uploaded Ticket class object.

a) The Web Service component calls the Query Event method (in the Query Event class) to query the related Event class object with the Event ID of the uploaded Ticket class object to get the detailed information of the Event.

b) The Web Service component decrypts the Event Signature of the uploaded Ticket class object with the Event Public Key of the related Event class object.

c) Compare the decrypted content with the Ticket ID and the Consumer ID. If they are not matched, the Web Service component responds to the Consumer Client component that the ticket is not real, and then the Consumer Client component responds to the Consumer user the same message. If they are matched, do the next steps.

Step 5: The Web Service component verifies the uploaded consumer signature.

a) The Web Service decrypts the uploaded consumer signature with the Consumer Public Key of the uploaded Ticket class object.

b) Compare the decrypted content with the Ticket Revoke Agreement Address of the related Event class object and the Ticket ID. If they are not matched, the Web Service component responds to the Consumer Client component that the consumer signature is not matched. If they are matched, do the next steps.

Step 6: The Web Service component accesses the event network and calls the Check If Ticket Is Revoked method (in the Check If Ticket Is Revoked class) on the Permissioned Blockchain component to check if the target ticket has already been revoked. If it is revoked, respond true and the related Ticket Revoke Record class object to the Web Service component, and then the Web Service component responds to the Consumer Client component a message that the ticket has already been revoked before with the related Ticket Revoke Record class object. The Consumer Client component responds to the

Consumer User that the ticket has already been revoked before. If it is not revoked, do the next steps.

Step 7: The Web Service component calls the Revoke Ticket method (in the Ticket Revoke class) to generate a Ticket Revoke Record class object with the uploaded Public Ticket Address, the consumer signature and the Ticket ID, and then uploads it to the event network on the Permissioned Blockchain component.

Step 8: The Web Service component accesses the event network, calls the Add Ticket Revoke Record method (in the Tickets Revoke List class) to add the uploaded Ticket Revoke Record class object into the related Tickets Revoke List class object, and calls the Remove Ticket Record method (in the Tickets List class) to remove the related ticket record from the Tickets List class object on the Permissioned Blockchain component.

Step 9: The Web Service component responds to the Consumer Client component the Ticket Revoke Record class object in Step 8 and a message that the target ticket has already been revoked.

Step 10: The Consumer Client component stores the responded Ticket Revoke Record class object in Step 9 and calls the Remove Ticket method (in the Consumer class) to remove the revoked ticket record from the Tickets List of the related Consumer class object.

During the ticket revocation process, because the data on the public blockchain cannot be deleted or modified, it is necessary to generate a revocation record to prove that the ticket has been revoked. In this process, the revocation record is in the form of a Ticket Revoke Record class object. The consumer signature in the record is to provide an undeniable evidence that the revocation of the target ticket has its consumer's consent so that the rights of the event party can be protected when a revoked ticket is used.

4.2.6 Event Registration

This section presents a work flow diagram to illustrate how an event is registered in the Hybrid Blockchain-Based Event Ticketing System as presented in Figure 4.19.

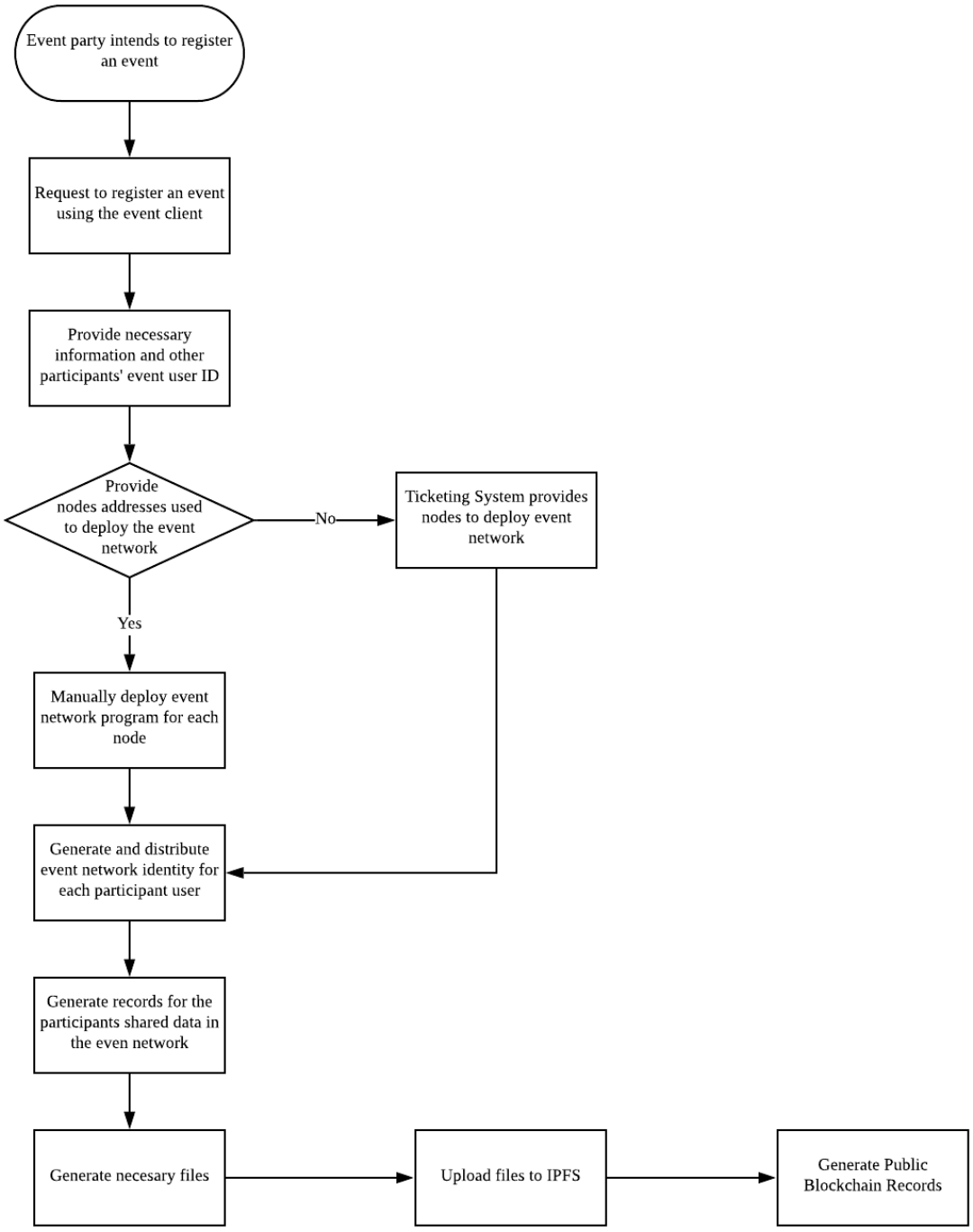


Figure 4.19 Work Flow of Event Registration

As can be seen in Figure 4.19, the work flow of the registration of an event can be described by the following steps.

Step 1: An Event Client user uses the Event Client to call the Register Event method (in the Event Client class) to request to register an event.

Step 2: The Event Client asks the user to provide other participants' event user ID and necessary information for the event, such as event location, event time, ticket purchase agreement, ticket revoke agreement and ticketing authorization consent etc. The user should indicate which information is shared data among the participants and which information is public data.

Step 3: The event user decides whether to provide the addresses of machines used to deploy the node program of the event network on the Permissioned Blockchain. If the user decides to provide them, the user needs to manually configure and deploy the related program to make the provided machines become the Permissioned Blockchain nodes of the event network. If the user does not provide, the user needs to provide the number of nodes, and the designed ticketing system will provide virtual machines and configure them as the Permissioned Blockchain nodes of the event network.

Step 4: Generate event network identities for the user and other participant users provided by the user in Step 2 to access the event network on the Permissioned Blockchain.

Step 5: Generate records for the participants shared event data provided by the user in Step 2 in the event network on the Permissioned Blockchain.

Step 6: The Web Service generates necessary files with the public data provided by the user in Step 2, such as the event information file.

Step 7: The Web Service uploads the generated files in Step 6 to the IPFS.

Step 8: The Web Service uses the addresses of the uploaded IPFS files in Step 6 to submit smart contracts to generate records on the Public Blockchain for the necessary files.

CHAPTER 5: EVALUATION

This chapter presents the experiments conducted on the main functions of the Hybrid Blockchain-Based Event Ticketing System, and presents the evaluation of the experimental results.

There are three experiments. Each of them is designed to evaluate a main function of the system. The evaluated functions are corresponding to the research problems. In addition to the experiments, this chapter also presents the cost calculation of the storage of the system. The summary of the content is shown as follows.

Content	Evaluated Function	Corresponding Research Problem
Experiment 1	Ticket Authenticity Evidence Generation	How to make tickets unforgeable?
Experiment 2	Ticket Record Generation and Storage	How to ensure the transparency of the ticketing data while protecting the privacy?
Cost Calculation		
Experiment 3	Ticket Entrance Verification	How to prevent ticket scalping?

Table 5.1 Summary of the Experiments

- **Experiment 1**

Experiment 1 is conducted on the implementation of the ticket authenticity evidence generation. The setup of Experiment 1 is shown in Figure 5.1.

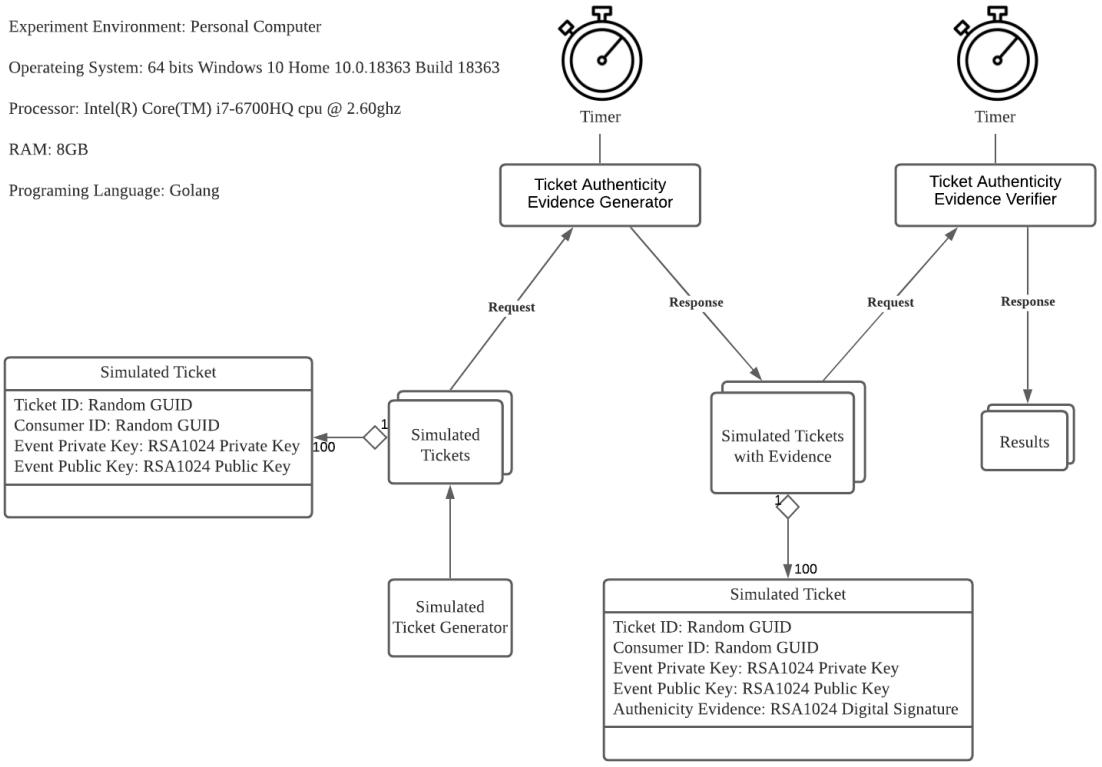


Figure 5.1 Setup of Experiment 1

As shown in Figure 5.1, in Experiment 1, there is a Simulated Ticket Generator program responsible for generating simulated tickets. A simulated ticket contains a Ticket ID which is a random GUID (Leach et al., 2005), a Consumer ID which is also a random GUID, and an event key pair generated by RSA1024 (Rivest et al., 1978). Each generated ticket is sent within a request to the Ticket Authenticity Evidence Generator program. The Ticket Authenticity Evidence Generator is responsible for generating an Authenticity Evidence for the received ticket and responding a simulated ticket data with the Evidence. Each generated simulated ticket with Evidence is sent within a request to the Ticket Authenticity Evidence Verifier program. The Ticket Authenticity Evidence Verifier is responsible for verifying the received ticket by decrypting the evidence and responding the verification results. There are two timers for recording the evidence generation time and the evidence verification time. Experiment 1 is conducted on a personal computer equipped with 64-bit windows 10 operating system, quad-core CPUs with 2.6GHz CPU clock speed and 8GB RAM. The programming language is Golang.

The research problem of how to make tickets unforgeable has been solved by the Hybrid Blockchain-Based Event Ticketing System using digital signature technology. As introduced in Chapter 4, in the system, each ticket has a digital signature signed by the private key of the related event with its ticket ID and consumer ID as evidence to ensure its authenticity. Experiment 1 is conducted to evaluate this solution. As a supplement to the setup, the specific steps of Experiment 1 are shown in Figure 5.2.

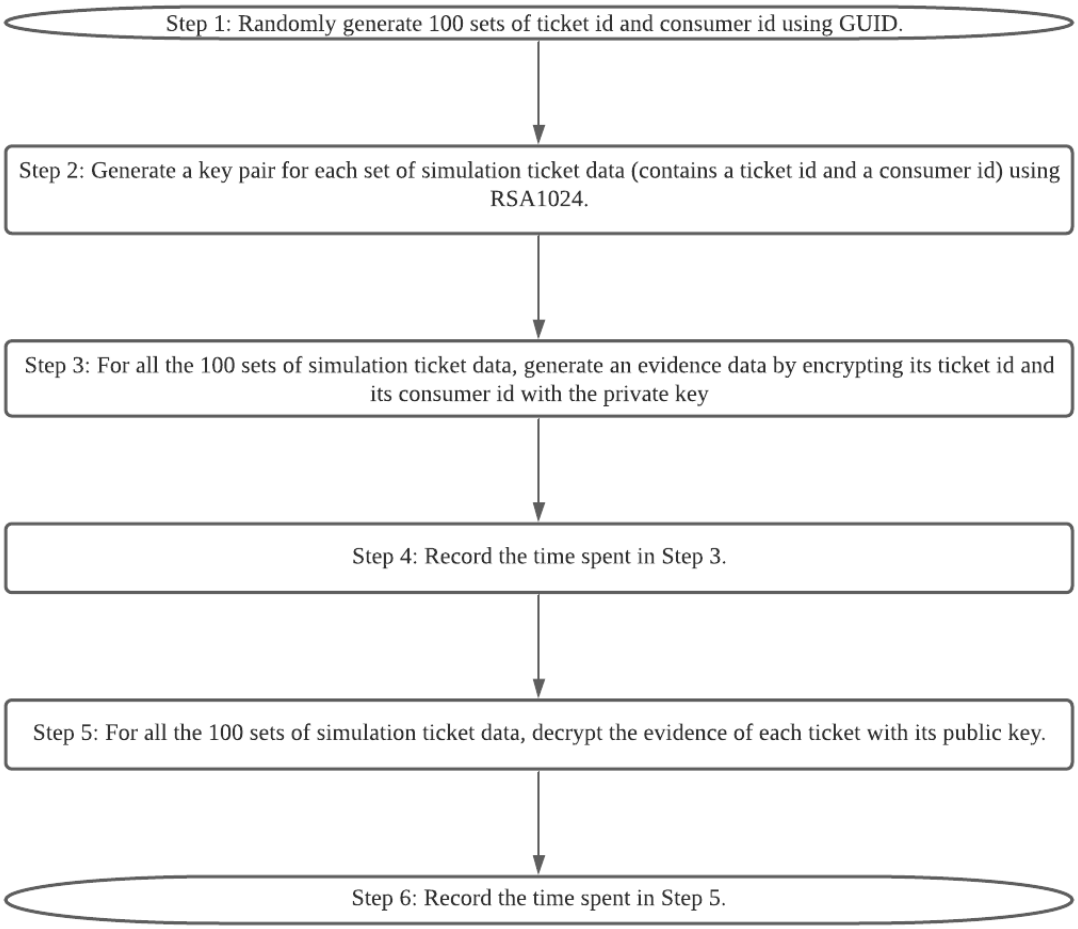


Figure 5.2 Steps of Experiment 1

As can be seen in Figure 5.2, in each experiment, the total time of generation and the total time of decryption of the 100 ticket authenticity evidence is recorded. This experiment is performed for 100 times.

The results of the total generation time of 100 ticket authenticity evidence in each experiment are shown in Figure 5.3.

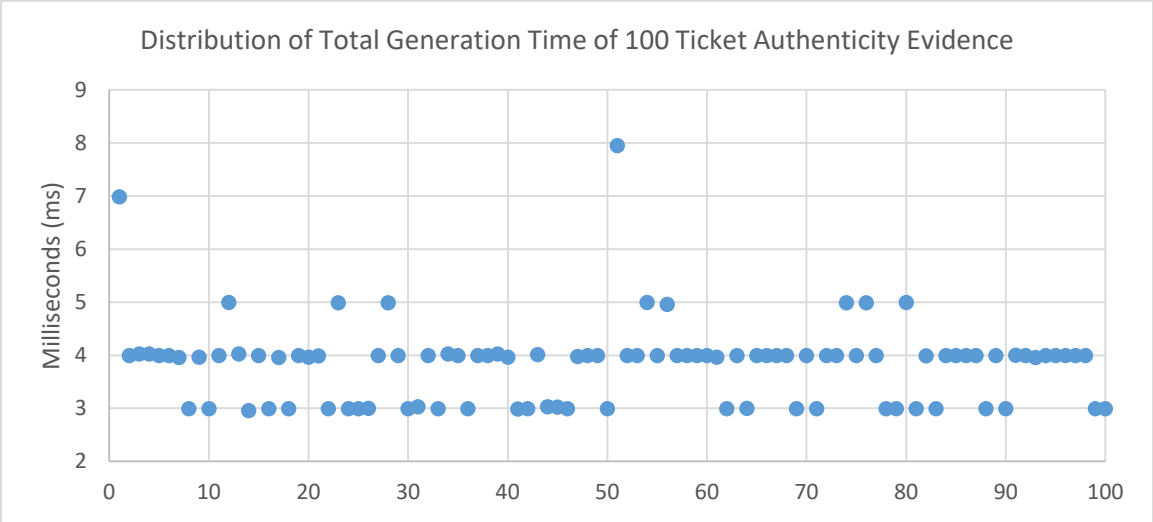


Figure 5.3 Distribution of Total Generation Time of 100 Ticket Authenticity Evidence

The arithmetic mean of the results in Figure 5.3 is 3.8296ms, while the median is 3.98925ms. Therefore, the arithmetic mean for a single time of the evidence generation is 0.038296ms, while the median is 0.0398935ms.

The results of the total decryption time of 100 ticket authenticity evidence in each experiment are shown in Figure 5.4.

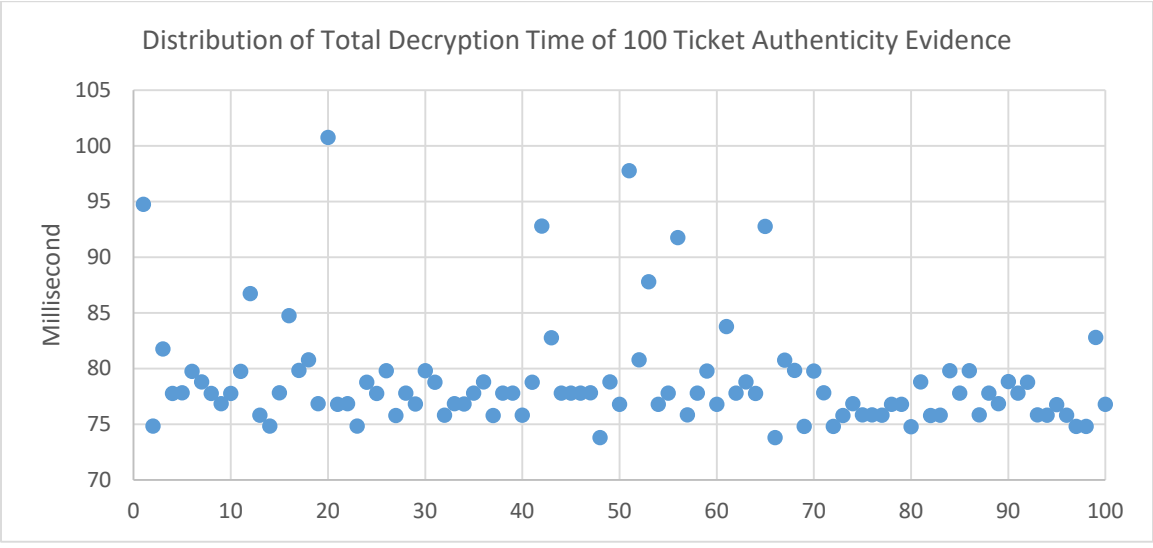


Figure 5.4 Distribution of Total Decryption Time of 100 Ticket Authenticity Evidence

The arithmetic mean of the results in Figure 5.4 is 78.87994ms, while the median is 77.789ms. Therefore, the arithmetic mean for a single time of the evidence decryption is 0.788799ms, while the median is 0.77789ms.

As can be seen from Figures 5.3 and 5.4, although most of the results are concentrated in the vicinity of the median and the arithmetic mean, there are still some results that differ greatly. One possible reason for these unusual results is that the experiment was conducted on a personal computer and thereby its extraneous environment, with additional unrelated background operating processes, may have interfered with the results. Executing a permissioned blockchain on a "clean" cloud-based virtual machine would be costly, beyond the author's research budget. The author's university lab's computer cluster would be a better target machine if its access was not prohibitive due to the 2020 University of Saskatchewan Covid-19 guidelines. Delaying the author's research until the end of the pandemic and the re-opening of the computer lab is not feasible. Hence, there are no options for the author except to conduct the experiment on a personal computer with external interference. In future work, Experiment 1 should be conducted using a cloud-based virtual machine or a computer cluster.

- **Experiment 2**

Experiment 2 is conducted on the implementation of the ticket record storage in the system. The setup of Experiment 2 is shown in Figure 5.5.

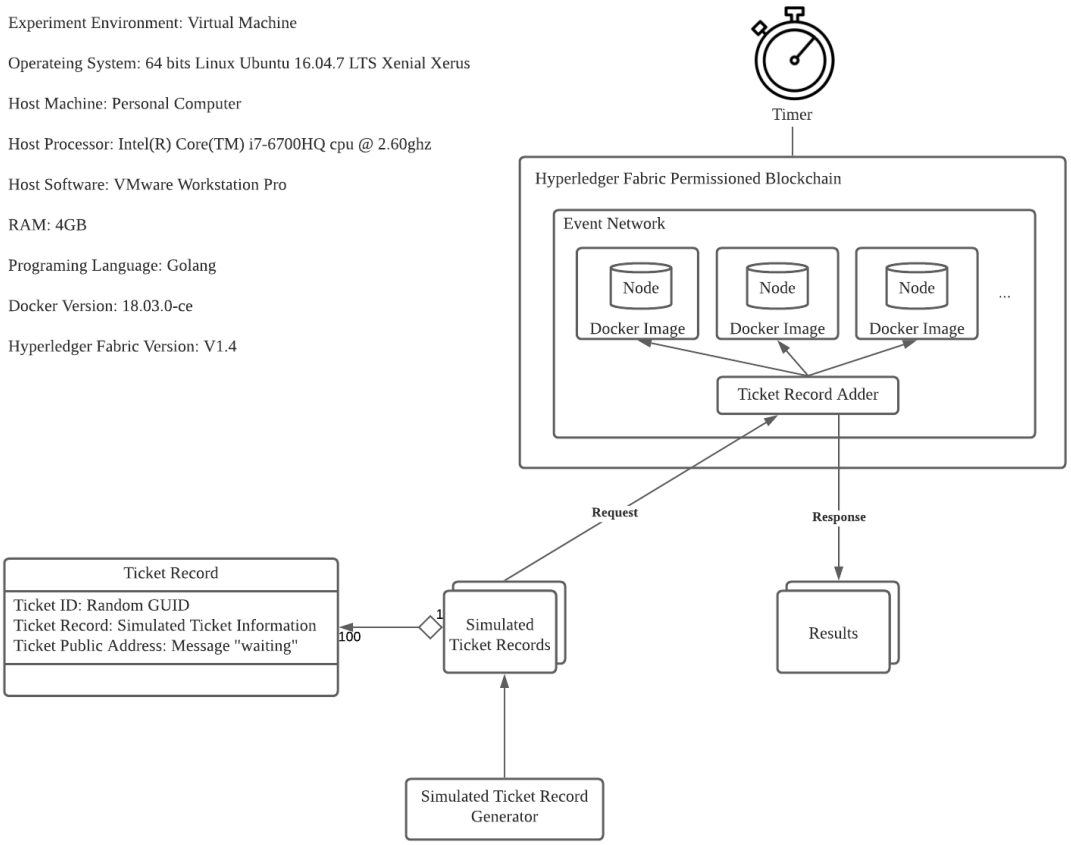


Figure 5.5 Setup of Experiment 2

As presented in Figure 5.5, in Experiment 2, there is a Simulated Ticket Record Generator program responsible for generating Simulated Ticket Records. A Simulated Ticket Record has a Ticket ID which is a Random GUID, a simulated Ticket Record which is a JSON string that contains simulated ticket information, and a Ticket Public Address which is a string “waiting”. Each Simulated Ticket Record is sent within a request to the Ticket Record Adder which is deployed in a permissioned blockchain-based event network. The permissioned blockchain is built within the frame of Hyperledger Fabric V1.4. The permissioned blockchain may have single node or multiple nodes according to the requirements. The Ticket Record Adder is responsible for determining whether to accept the received ticket record or not, according to the remaining ticket number. If the remaining ticket number is greater than 0, the Ticket Record Adder will record the received ticket and minus the remaining ticket number by one, and then respond the result. There is also a timer for recording the time cost by the Ticket Record Adder to process ticket records.

Experiment 2 is conducted on a virtual machine with 64 bits Linux Ubuntu operating system and 4GB RAM. The virtual machine is operated by the VMware Workstation. It is hosted by a personal computer. The host computer is equipped with 64-bit Windows 10 operating system and 8GB RAM. The permissioned blockchain is built within the frame of the Hyperledger Fabric 1.4. The nodes of the event network are executed in Docker Image.

The research problem of how to ensure the transparency of the ticketing data while protecting the privacy of private data was solved by the system with a hybrid storage architecture that uses permissioned blockchain and public blockchain to separately store data with different requirements of transparency and privacy. In this solution, the number of the remaining tickets for an event is the resource that can only be occupied and modified by one thread at the same time. Each thread has to wait for the previous thread to release this resource before it can execute, which is the major factor that limits the maximum throughput of the ticket record generation in addition to the performance of hardware. Experiment 2 is conducted to evaluate the throughput of the solution on this point. As a supplement to the setup, the specific steps of Experiment 2 are shown in Figure 5.6.

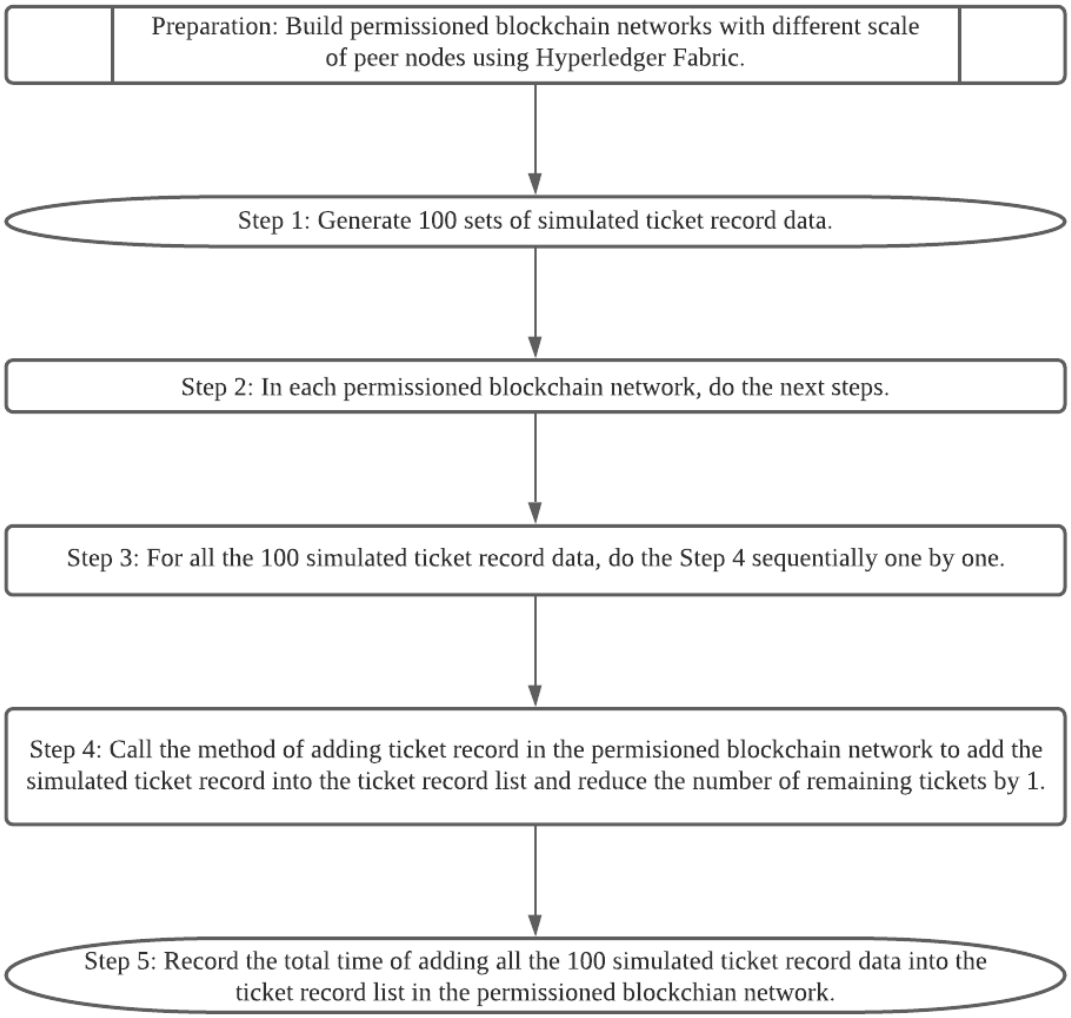


Figure 5.6 Steps of Experiment 2

As shown in Figure 5.6, the experiments are conducted in the permissioned blockchain networks with different scales of peer nodes. The throughput is calculated by the total time to uploading 100 simulated ticket record data.

The results of the experiments are shown in Table 5.2.

Peer Nodes Scale	Time Spent	Throughput
1 Peer	7 minutes and 20.38 seconds	13 - 14 (13.625) Transactions Per Minute
2 Peers	7 minutes and 35.78 seconds	13 - 14 (13.164) Transactions Per Minute
3 Peers	7 minutes and 49.37 seconds	12 - 13 (12.784) Transactions Per Minute
4 Peers	8 minutes and 21.60 seconds	11 - 12 (11.962) Transactions Per Minute
5 Peers	8 minutes and 58.54 seconds	11 - 12 (11.141) Transactions Per Minute
6 Peers	9 minutes and 51.75 seconds	10 - 11 (10.139) Transactions Per Minute
7 Peers	11 minutes and 55.31 seconds	8 - 9 (8.388) Transactions Per Minute

Table 5.2 Throughput of the Ticket Generation

The data in Table 5.2 can be displayed visually using a chart as shown in Figure 5.7.

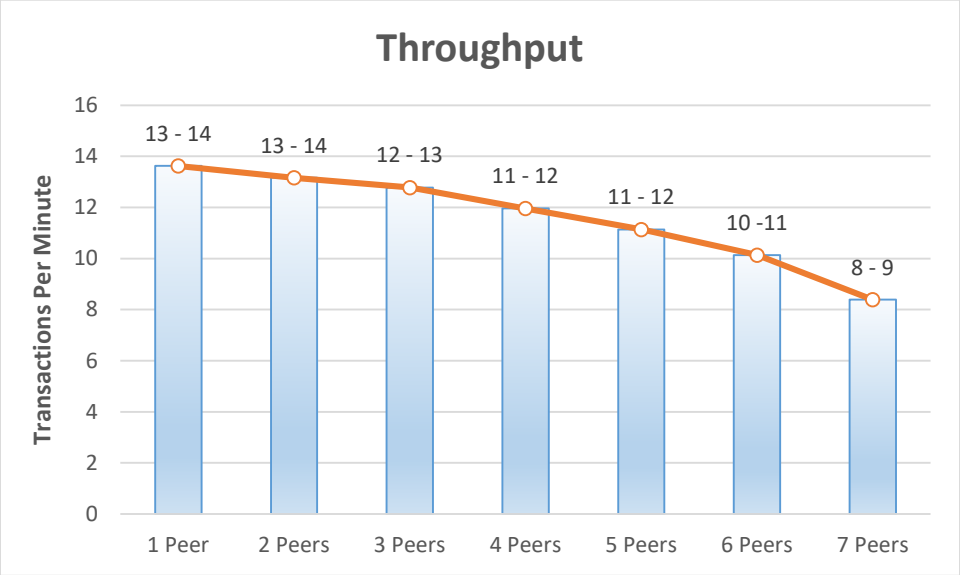


Figure 5.7 Chart of Throughput of Ticket Generation

As can be seen in Figure 5.7, the more peer nodes deployed in the event network of the permissioned blockchain, the less the throughput of ticket generation. This is caused by the increase in communications between the nodes. To improve the throughput of the ticket generation, while not affecting the features of the system design, but at the same time improving hardware performance, one option is to explore more efficient implementation of the permissioned blockchain instead of using Hyperledger Fabric. As explained, with regards to Experiment 1's results, the author believes the results of Experiment 2 were also affected due to the execution on a personal computer with its extraneous environment of additional unrelated background processes. In the future, this external interference could

be reduced by conducting the experiment on a cloud-based virtual machine or computer cluster. However, due to the unattainable high cost of availing of the resources of a cloud-based virtual machine, and having no access to the computer lab cluster, these required special processing needs could not be accommodated.

- **Cost Calculation of the Ticket Record Storage**

Due to the global scale and the computing power based consensus algorithms, storing data on a public blockchain is expensive. Therefore, for the Hybrid Blockchain-Based Event Ticketing System, the major cost is the ticket records stored on the public blockchain.

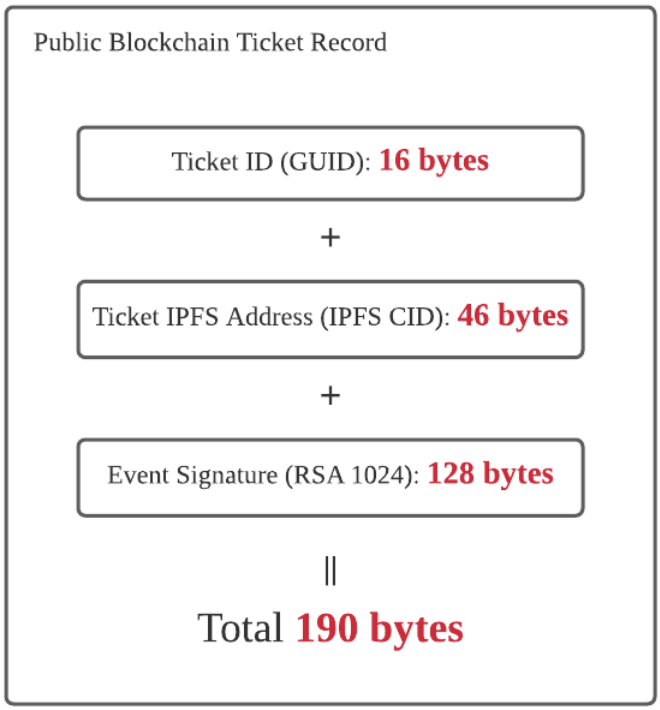


Figure 5.8 Size of a Ticket Record in Public Blockchain Part

As shown in Figure 5.8, in the system, the public blockchain ticket record is designed to consist of three necessary data, which are the ticket ID, the address of the IPFS file of the ticket and the event signature. This research used the GUID method to generate ticket ID. Therefore, the size of the ticket ID of each ticket is 16 bytes (GUID consists of 128-bit numbers, so the size of a GUID is 16 bytes (Leach et al., 2005)). An IPFS address (CID) is a 46 character string by default, so the size of the address of the IPFS file of the ticket is

46 bytes. The experiment used RSA1024 (Rivest et al., 1978) as the key pair (public key and private key) generation algorithm, and the length of the ciphertext encrypted by the private key is the same as the length of the private key, which is 1024 bits. Thus, the size of the event signature is 128 bytes. Therefore, the total size of a public blockchain ticket record is 16 bytes + 46 bytes + 128 bytes = 190 bytes.

The experiment implements the public blockchain component using Ethereum. Storing data on the Ethereum public blockchain costs gas. According to the Ethereum project yellow paper introduced by Wood (2014), it costs 20000 gas for storing a word which is 32 bytes of data in the Ethereum. Therefore, for storing a public blockchain ticket record of the system, which is at a total of 190 bytes, it will cost 190 / 32 *20000 = 118750 gas. By the time of September 2020, the gas price is $7*10^{10}$ Wei, 1 Ether = 1,000,000,000,000,000,000 Wei (10^{18}), 118750 gas will cost $118750 * 7 * 10^{10}/10^{18}$ = 0.0083125 Ether. The price of Ether is 343.40 USD. Thus, the cost for storing a public blockchain ticket record of the system in Ethereum is 0.0083125 * 343.40 = 2.8545125 ≈ 2.85 USD.

- **Experiment 3**

Experiment 3 was conducted on the implementation of the ticket record storage in the system. The setup of Experiment 3 is shown in Figure 5.9.

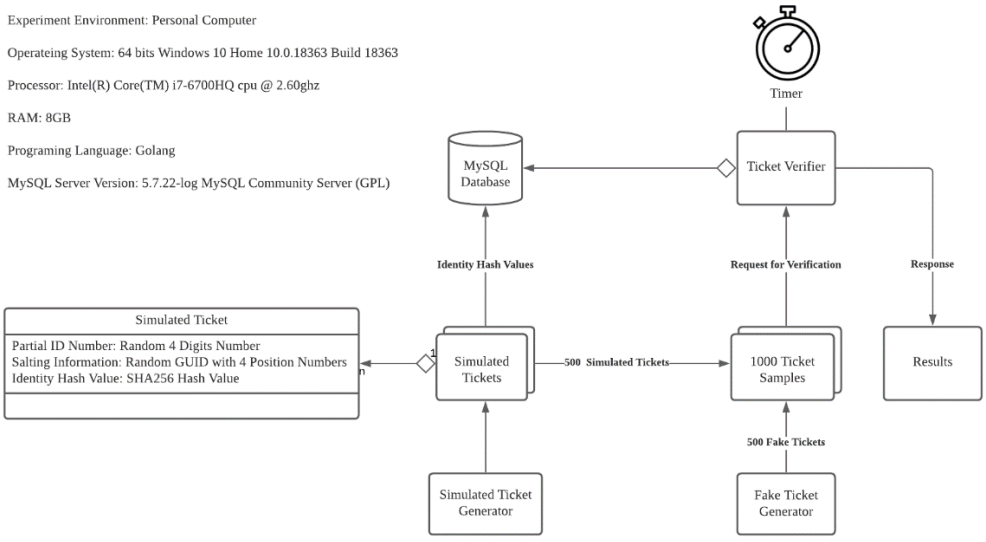


Figure 5.9 Setup of Experiment 3

As shown in Figure 5.9, in Experiment 3, there is a Simulated Ticket Generator responsible for generating Simulated Tickets. A simulated ticket has a Partial ID number which is a random 4-digit number, a Salting Information which is a string consisting of a random GUID and 4 position numbers between 0 and 31, and an Identity Hash Value generated with SHA256 hash method (National Security Agency, 2001). The Identity Hash Value of the generated tickets is stored in a MySQL database. There is also a Fake Ticket Generator responsible for generating fake tickets whose identity hash values are not stored in the database. There are 1000 ticket samples including 500 of the generated tickets whose identity hash values are recorded in the database and 500 fake tickets. Each ticket sample is sent within a request to the Ticket Verifier program for verification. The Ticket Verifier verifies whether the received ticket's identity hash value is included in the database or not, and then responds the result. There is also a Timer responsible for recording the time taken by the Ticket Verifier to verify tickets. Experiment 3 is conducted on a personal computer equipped with 64-bit windows 10 operating system, quad-core CPUs with 2.6GHz CPU clock speed and 8GB RAM. The programming language is Golang. The MySQL server version is 5.7.22-log MySQL Community Server (GPL).

The research problem of scalping prevention is solved by the system using a method of generating an identity proof for each ticket. The identity proof is an encrypted hash value of a salted string of ticket consumer's partial ID number. Experiment 3 is conducted to evaluate this solution. As a supplement, in the experiment, the ticket data with an identity proof is generated with the following specific steps as shown in Figure 5.10.

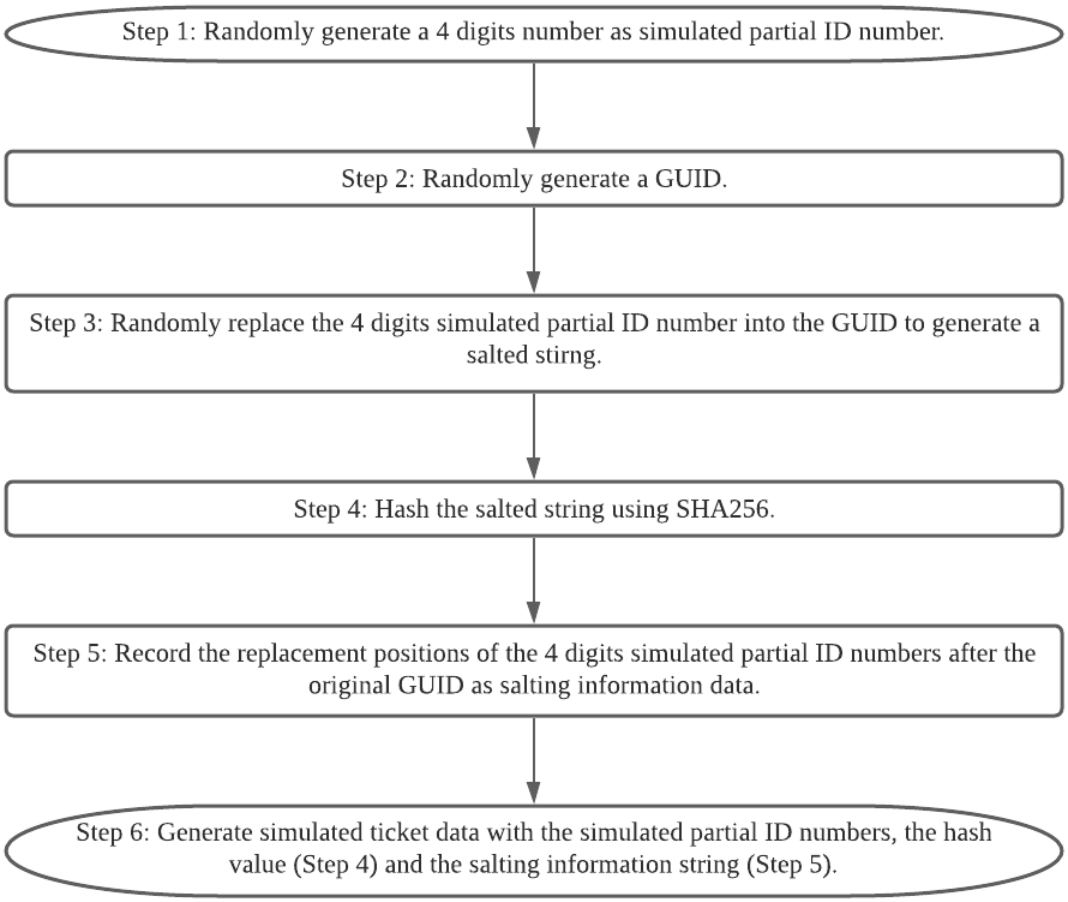


Figure 5.10 Generation of the Ticket Identity Proof in Experiment 3

And the specific steps of Experiment 3 are shown in Figure 5.11.

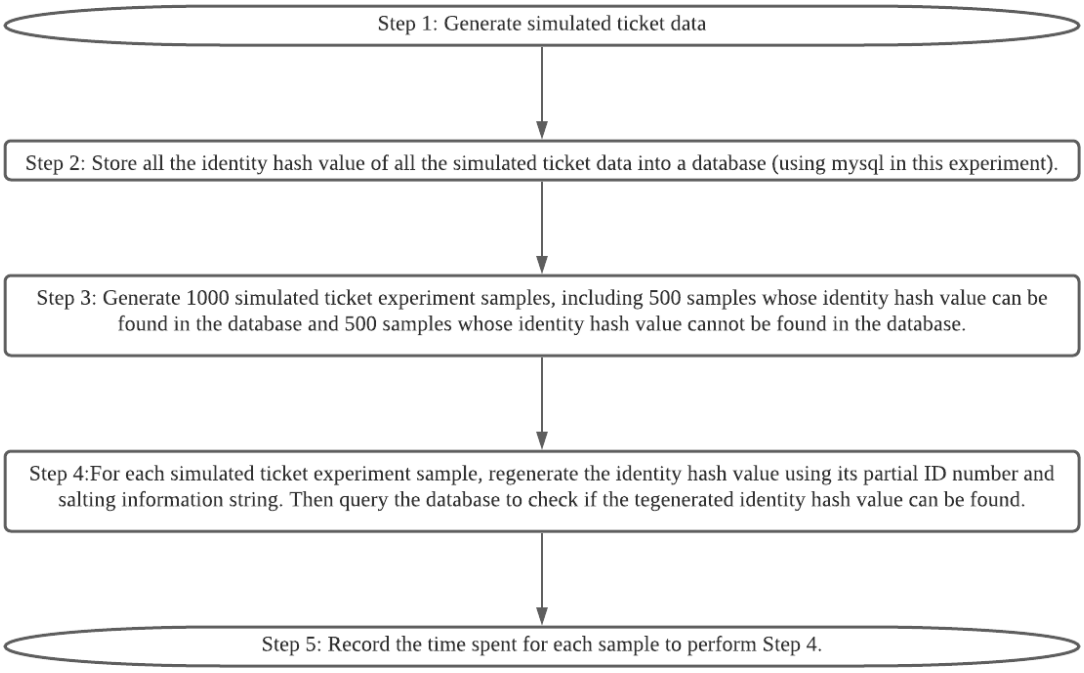


Figure 5.11 Steps of Experiment 3

As shown in Figure 5.11, Experiment 3 is conducted with different scales of data sets. This is because the scale of the data set influences the speed of querying. It is necessary to compare and evaluate the efficiency of ticket entrance verification under different data scales. Experiment 3 evaluated the efficiency of ticket entrance verification by the response time of a single entrance ticket verification process.

For the event scale of 500 tickets, such as events in small theaters or cinemas, the response time distribution of the 1000 tested ticket verification samples (including 500 real tickets and 500 fake tickets) is shown in Figure 5.12.

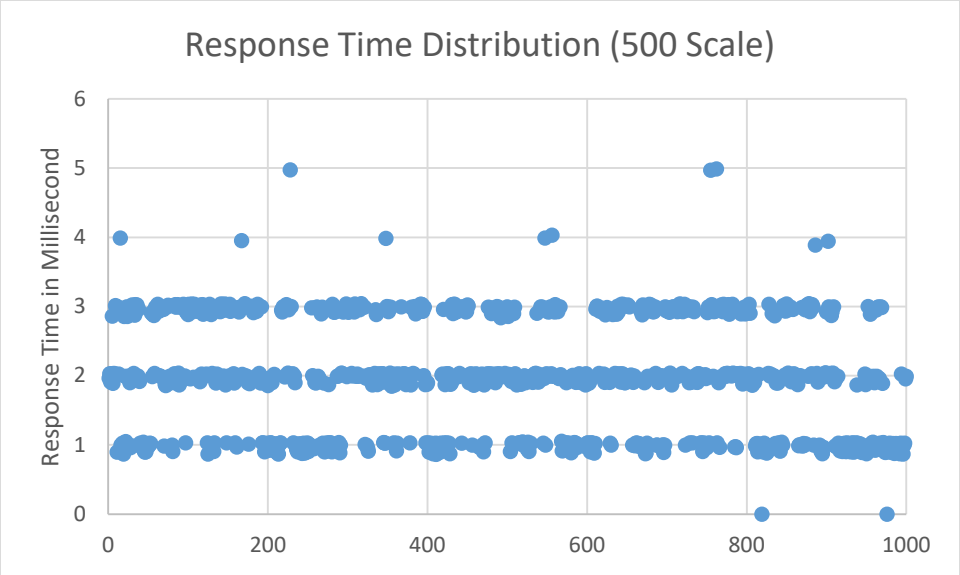


Figure 5.12 Response Time Distribution for Event Scale of 500 Tickets

For the event scale of 1,500 tickets, such as events in large theaters, the response time distribution of the 1000 tested ticket verification samples (including 500 real tickets and 500 fake tickets) is shown in Figures 5.13.

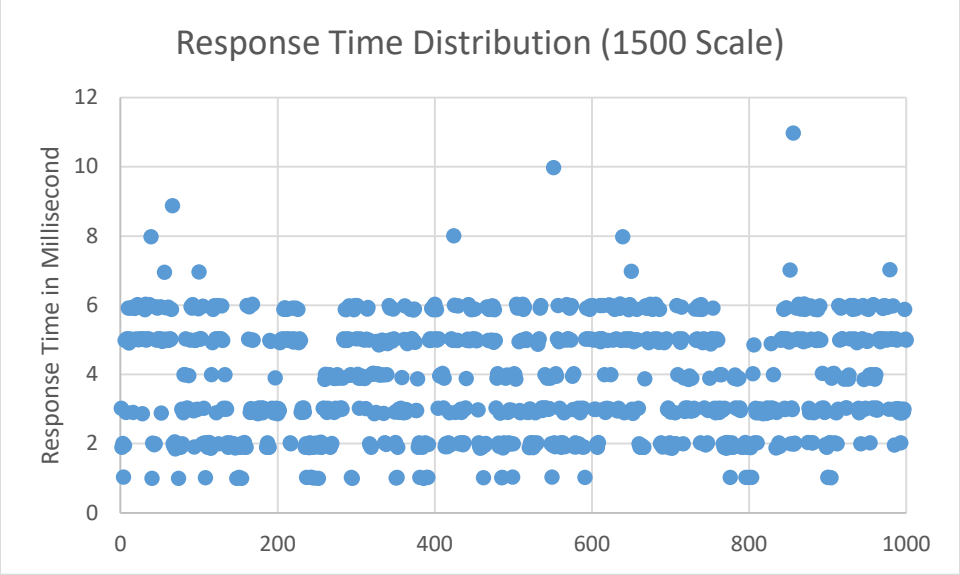


Figure 5.13 Response Time Distribution for Event Scale of 1,500 Tickets

For the event scale of 5,000 tickets, such as a concert or a medium-sized sports events (a volleyball match), the response time distribution of the 1000 tested ticket verification samples (including 500 real tickets and 500 fake tickets) is shown in Figure 5.14.

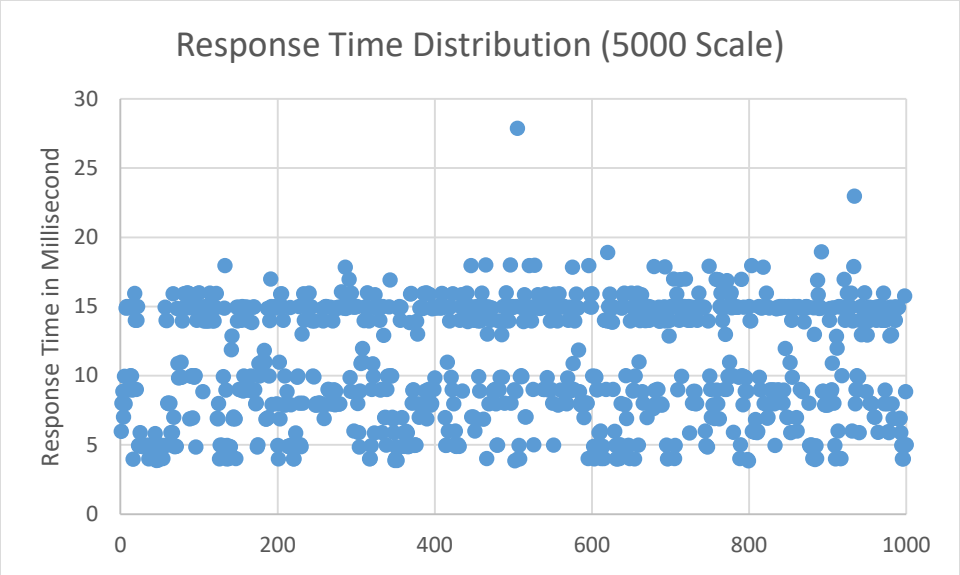


Figure 5.14 Response Time Distribution for Event Scale of 5,000 Tickets

For the event scale of 15,000 tickets, such as a medium-sized concert or a major sports event (a major basketball game), the response time distribution of the 1000 tested ticket verification samples (including 500 real tickets and 500 fake tickets) is shown in Figure 5.15.

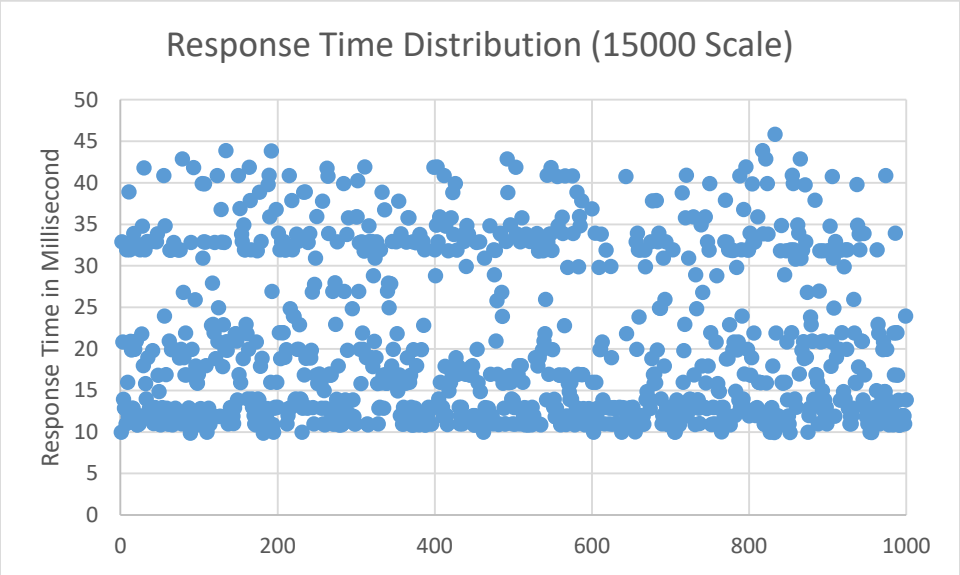


Figure 5.15 Response Time Distribution for Event Scale of 15,000 Tickets

For the event scale of 50,000 tickets, such as a mega concert or a mega sports event (a major football game), the response time distribution of the 1000 tested ticket verification samples (including 500 real tickets and 500 fake tickets) is shown in Figure 5.16.

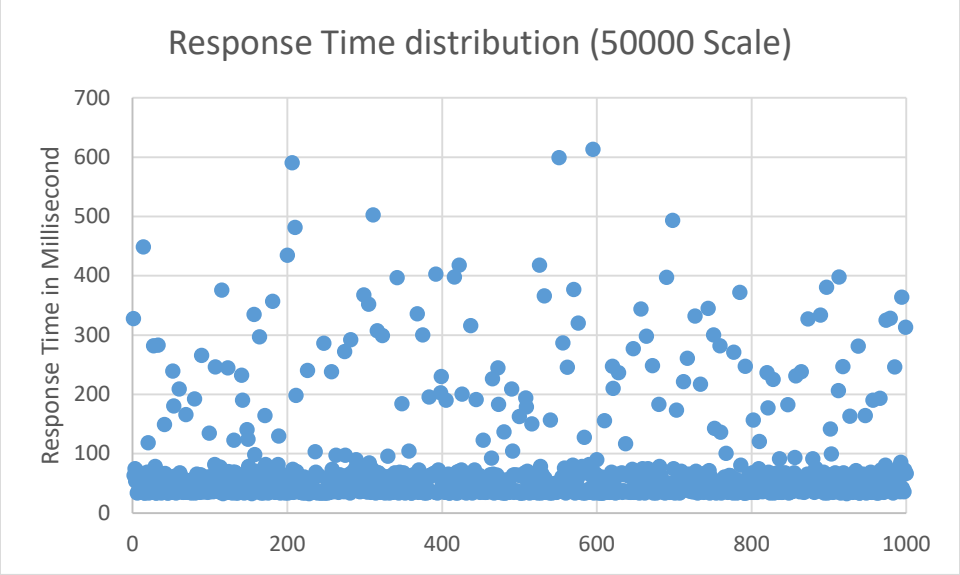


Figure 5.16 Response Time Distribution for Event Scale of 50,000 Tickets

For the event scale of 150,000 tickets, such as a mega music festival, the response time distribution of the 1000 tested ticket verification samples (including 500 real tickets and 500 fake tickets) is shown in Figure 5.17.

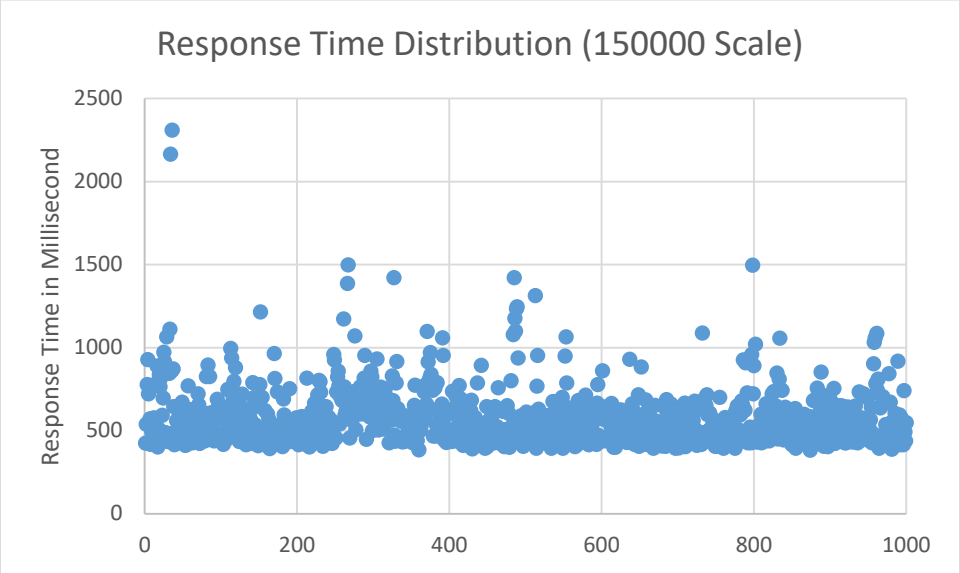


Figure 5.17 Response Time Distribution for Event Scale of 150,000 Tickets

The arithmetic mean and median of the ticket verification response time for each event scale are summarized in Table 5.3 as follows.

Event Scale (Tickets)	Arithmetic Mean of Response Time (Millisecond)	Median of Response Time (Millisecond)
500	1.9981195ms	1.9934ms
1500	3.8036905ms	3.8802ms
5000	11.0766713ms	10.9762ms
15000	20.3172742ms	16.8215ms
50000	75.5780024ms	50.8286ms
150000	573.3602ms	526.583ms

Table 5.3 Arithmetic Means and Medians of the Ticket Verification Response Time Table

The data in Table 5.3 can be displayed visually using a column chart as shown in Figure 5.18.

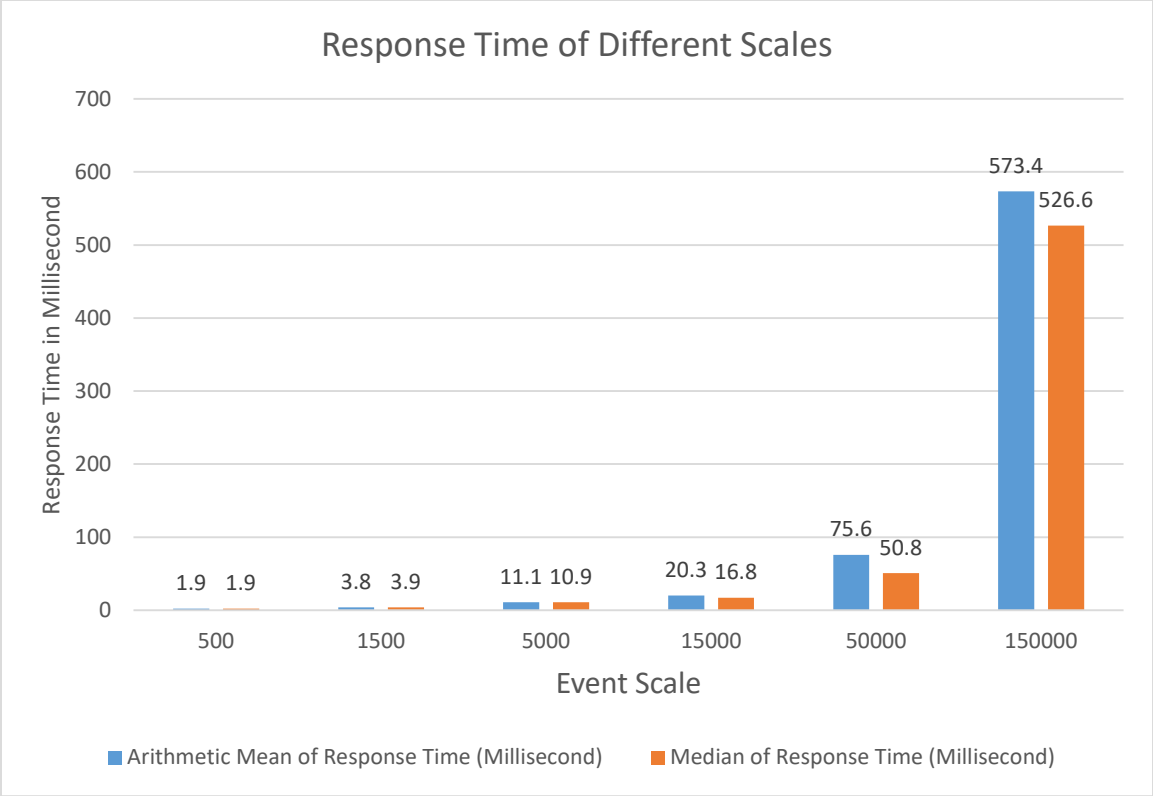


Figure 5.18 Arithmetic Means and Medians of the Ticket Verification Response Time Chart

As can be seen in Figure 5.18, the response time of ticket verification increases with the scale of the event. The median response time of ticket verification for an event of the scale of 50,000 tickets is 50.8 milliseconds. When the scale reaches 150,000 tickets, this data increases by more than 10 times to 526.6 milliseconds. This is because the larger the data set, the more time it takes for scanning the data set, and the longer the response time. If the number of tickets of an event is too large, it can consider designating an entrance for each ticket to divert audience. Each ticket is only stored in the ticket verification system of its corresponding entrance. In this way, a large data set will be divided into several different subsets and distributed at each entrance ticket verification system. Therefore, the size of the data set for a single entrance is reduced, resulting in the reduction of scanning time, thereby reducing the response time of ticket verification. However, the author did not conduct experiments to implement the optimization due to time constraints.

According to the results presented in Figures 5.12, 5.13, 5.14, 5.15, 5.16 and 5.17, at every scale, there are some results that are far from the median and the arithmetic mean. The

reason for these unusual results might be due to the experiment being conducted on a personal computer, and thereby the background interference from the external environment could be huge. It can be seen that the larger the scale, the more concentrated the distribution of the results. This may be because the larger the data scale, the longer the time required to query the database in the verification, and the smaller the proportion of the time, as influenced by the external interference. The external interference can be reduced by conducting the experiment on a cloud-based virtual machine or computer cluster. However, due to the unattainable high cost of availing of the resources of a cloud-based virtual machine, and having no access to the computer lab cluster, these required special processing needs could not be accommodated.

CHAPTER 6: CONCLUSION

This chapter gives a general conclusion of the research and introduces future work.

6.1 Conclusion

To solve the problems of forged ticket prevention, privacy protection, transparency and traceability of ticketing information, as well as scalping prevention, this thesis introduces a hybrid blockchain-based ticketing system. The event ticketing system is a complex information system, in which various information is shared with different groups and scopes, so that different information has different requirements for transparency, privacy and information sharing policy. Therefore, the information in the system should be subdivided into different categories according to its target user groups. And different categories of information should be stored in different infrastructures to suit their types. In this regard, the system contains a permissioned blockchain and a public blockchain. The permissioned blockchain is used to store non-public event ticketing information that should only be shared among event participants, such as the information of ticket sales and agreements signed between event participants. The public blockchain is used to store public information that should be transparent to the public and should not be able to be unilaterally changed or deleted, such as ticket purchase agreements and public event information. The public blockchain is also used to record tickets to ensure that every ticket can be tracked by its owner. The record cannot be changed or deleted. Thus, the public blockchain ticket record can be used as evidence to ensure the protection of consumer rights when there is a dispute. To reduce the cost of storing data on public blockchain, the system uses IPFS files (Benet, 2014) to store the detailed information of each ticket, and only stores the address of the IPFS files on the public blockchain. This research also designs a specific method for the system to digitally sign each ticket to ensure that the tickets cannot be forged and its authenticity can be verified with public information. In the case that the information of each ticket is public and transparent, all the non-public information stored on the tickets is encrypted to protect the privacy of users. The permissioned blockchain part, public blockchain part and other components in the system interact with each other through a web service component to ensure the consistency of the information. The non-public information transmitted in the web service component is also encrypted before uploading

to protect privacy. In addition, the system also includes an entrance ticket verification mechanism to prevent scalping while protecting the privacy of consumers. The designed mechanism increases the cost and risk of ticket scalping greatly while keeping the real identity of the consumer uncertain by forming a many-to-one relationship between the real identity of the consumer and the identity proof recorded on the ticket. Specifically, the implementation uses the hash value of the salted partial consumer's real identity number as the ticket owner's identity proof recorded on the ticket. The salting information is retained by the consumer until it is provided with the consumer's partial real identity number during the entrance ticket verification to regenerate the ticket owner's identity proof to be compared with the record on the ticket stored in the system. In this way, only the ticket owner's identity proof data is stored in the system, and real identity of the consumer is impossible to reverse from the identity proof without knowing the salting information, which is only kept by the consumer. The system architecture is abstract. It is not limited to be implemented with specific technologies.

6.2 Future Work

Although the Hybrid Blockchain-Based Event Ticketing System provided solutions to the research problems. There are still some problems to be discussed and researched in future work.

1. The concurrency of the ticket generation needs to be improved. In the system, limited by the blockchain-based storage structure, ticket generation cannot be completed with high concurrency.

2. The cost of data storage needs to be further reduced.

3. The specific implementation of the system can be further studied. The presented system architecture is abstract. In future research, specific implementation schemes can be compared and evaluated from different aspects to find the most suitable one for different application scenarios.

4. Experiments need to be conducted under the simulation of real scenarios. In the real application scenario, different components, nodes are deployed in different places. The

communications between them are remote. The delay that is caused by the net connection can greatly affect the results of the experiments.

5. The full life cycle of ticket generation needs to be explored and evaluated. Limited by funding and time, this study did not conduct experiments on the complete life cycle of the ticket generation process.

6. The experiments need to be conducted using larger-scale data sets to better represent real scenarios.

7. The experiments need to be conducted on a cloud-based virtual machine or a computer cluster to reduce the influence of the external interference.

REFERENCES

Aventus Protocal Foundation. (2018). A blockchain-based event ticketing protocol [White paper]. Retrieved March 18, 2019, from Aventus: https://3.9.12.207/wp-content/uploads/2019/03/Whitepaper.pdf

Becker, G. (2008). Merkle signature schemes, merkle trees and their cryptanalysis. Ruhr-University Bochum, Tech. Rep.

Bell, J. (2005). Ticket scalping: Same old problem with a brand new twist. Loy. Consumer L. Rev., 18, 435.

Benet, J. (2014). Ipfs-content addressed, versioned, p2p file system. arXiv preprint arXiv:1407.3561.

Buterin, V. (2016). Ethereum platform review: Opportunities and challenges for private and consortium blockchains. Retrieved March 2, 2021, from internet: http://www.smallake.kr/wp-content/uploads/2016/06/314477721-Ethereum-Platform-Review-Opportunities-and-Challenges-for-Private-and-Consortium-Blockchains.pdf

Cachin, C. (2016, July). Architecture of the hyperledger blockchain fabric. In Workshop on distributed cryptocurrencies and consensus ledgers (Vol. 310, No. 4).

Castro, M., & Liskov, B. (1999). Practical Byzantine fault tolerance. In OSDI (Vol. 99, No. 1999, pp. 173-186).

Cha, S. C., Peng, W. C., Hsu, T. Y., Chang, C. L., & Li, S. W. (2018). A Blockchain-Based Privacy Preserving Ticketing Service. In 2018 IEEE 7th Global Conference on Consumer Electronics (GCCE) (pp. 585-587). IEEE.

Chung, K., Yoo, H., Choe, D., & Jung, H. (2019). Blockchain network based topic mining process for cognitive manufacturing. Wireless Personal Communications, 105(2), 583-597.

Dai, W. (1998). B-money. Consulted, 1, 2012.

Diffie, W., & Hellman, M. (1976). New directions in cryptography. IEEE transactions on Information Theory, 22(6), 644-654.

Eastlake, D., & Jones, P. (2001). US secure hash algorithm 1 (SHA1). Retrieved March 2, 2021, from internet: https://www.hjp.at/(st_a)/doc/rfc/rfc3174.html

Finney, H. (2004). Rpow-reusable proofs of work. Retrieved March 2, 2021, from internet: https://cryptome. org/rpow. htm.

Finžgar, L., & Trebar, M. (2011). Use of NFC and QR code identification in an electronic ticket system for public transport. In SoftCOM 2011, 19th International Conference on Software, Telecommunications and Computer Networks (pp. 1-6). IEEE.

GET Foundation Team. (2017). Guaranteed entrance token smart event ticketing protocol [White paper]. Retrieved March 18, 2019, from GUTS Tickets: https://guts.tickets/files/GET-Whitepaper-GUTS-Tickets-latest.pdf

Göbel, J., & Krzesinski, A. E. (2017). Increased block size and Bitcoin blockchain dynamics. In 2017 27th International Telecommunication Networks and Applications Conference (ITNAC) (pp. 1-6). IEEE.

Hao, F. (2017). Schnorr non-interactive zero-knowledge proof. RFC 8235, Sept.

Isaksson, C., & Elmgren, G. (2018). A ticket to blockchains. Retrieved March 2, 2021, from internet: https://www.diva-portal.org/smash/get/diva2:1282090/FULLTEXT02

Jeppsson, A., & Olsson, O. (2017). Blockchains as a solution for traceability and transparency. Retrieved March 2, 2021, from internet: https://lup.lub.lu.se/student-papers/search/publication/8919957

King, S., & Nadal, S. (2012). Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. self-published paper, August, 19, 1. Retrieved March 2, 2021, from internet: https://www.chainwhy.com/upload/default/20180619/126a057fef926dc286accb372da46955.pdf

Ko, D. H., Choi, H. K., & Kim, K. S. (2020). A design and implementation of macro prevention ticket booking system using blockchain. In Proceedings of the 2020, the 6th International Conference on E-Business and Applications (pp. 95-98).

Kuznetsov, A., Svatovskij, I., Kiyan, N., & Pushkar'ov, A. (2017). Code-based public-key cryptosystems for the post-quantum period. In 2017 4th International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T) (pp. 125-130). IEEE.

Lamport, L. (1983). The weak Byzantine generals problem. Journal of the ACM (JACM), 30(3), 668-676.

Lamport, L. (2001). Paxos made simple. ACM Sigact News, 32(4), 18-25.

Larimer, D. (2014). Delegated proof-of-stake (dpos). Bitshare whitepaper, 81, 85. Retrieved March 2, 2021, from internet: https://whitepaper.io/document/388/bitshares-whitepaper

Leach, P., Mealling, M., & Salz, R. (2005). A universally unique identifier (uuid) urn namespace. Retrieved March 2, 2021, from internet: https://www.hjp.at/doc/rfc/rfc4122.html

Leslie, L. (1998). The part-time parliament. ACM Transactions on Computer Systems, 16(2), 133-169.

Lin, K. P., Chang, Y. W., Wei, Z. H., Shen, C. Y., & Chang, M. Y. (2019). A Smart Contract-Based Mobile Ticketing System with Multi-Signature and Blockchain. In 2019 IEEE 8th Global Conference on Consumer Electronics (GCCE) (pp. 231-232). IEEE.

Mingxiao, D., Xiaofeng, M., Zhe, Z., Xiangwei, W., & Qijun, C. (2017). A review on consensus algorithm of blockchain. In 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC) (pp. 2567-2572). IEEE.

Morris, R., & Thompson, K. (1979). Password security: A case history. Communications of the ACM, 22(11), 594-597.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. Retrieved March 2, 2021, from internet: https://git.dhimmel.com/bitcoin-whitepaper

Nakamoto, S. (2009). Base58. h. Digital. Accessed 2020-02-20. URL: https://github.com/bitcoin/bitcoin/blob/aaaaad6ac95b402fe18d019d67897ced6b316ee0/src/base58. h.

Oechslin, P. (2003). Making a faster cryptanalytic time-memory trade-off. In Annual International Cryptology Conference (pp. 617-630). Springer, Berlin, Heidelberg.

Ongaro, D., & Ousterhout, J. (2017). In Search of an Understandable Consensus Algorithm.

Qteishat, M. K., Alshibly, H. H., & Al-ma'aitah, M. A. (2014). The impact of e-ticketing technique on customer satisfaction: an empirical analysis. JISTEM-Journal of Information Systems and Technology Management, 11(3), 519-532.

Peck, M. E. (2017). Blockchain world-Do you need a blockchain? This chart will tell you if the technology can solve your problem. IEEE Spectrum, 54(10), 38-60.

Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2), 120-126.

Tackmann, B. (2017). Secure Event Tickets on a Blockchain. In Data Privacy Management, Cryptocurrencies and Blockchain Technology (pp. 437-444). Springer, Cham

Vitalik, B. (2013). Ethereum white paper: a next generation smart contract & decentralized application platform. Retrieved March 2, 2021, from internet: https://cryptorating.eu/whitepapers/Ethereum/Ethereum_white_paper.pdf

Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper, 151(2014), 1-32.

Wüst, K., & Gervais, A. (2018). Do you need a blockchain?. In 2018 Crypto Valley Conference on Blockchain Technology (CVCBT) (pp. 45-54). IEEE.