A Blockchain-based Privacy-preserving Medical Insurance Storage System

by

Son Luong

Bachelor of Information System, Hanoi University of Science and Technology, 2013

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Computer Science

In the Graduate Academic Unit of Computer Science

Supervisor(s):	Rongxing Lu, Ph.D, Faculty of Computer Science
Examining Board:	Dima Alhadidi, Ph.D, Faculty of Computer Science, Chair Suprio Ray, Ph.D, Faculty of Computer Science, Gouhua Yan, Ph.D, Department of Mathematics and Statistics

This thesis is accepted

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

April, 2019

Abstract

Blockchain technology is an innovative invention that is disrupting many industries including business and healthcare. In this thesis, we propose a blockchain-based privacy-preserving medical insurance storage system. This system takes advantage of the decentralization and immutability properties of blockchain technology, and makes use of a (2,3)-threshold secret sharing scheme to achieve the privacy-preservation property. In an experimental setup of the system, there are a public blockchain, a patient, four hospitals - an owner hospital and three helper hospitals, and an insurance company. For a patient, any hospital can become the owner hospital while the other three become helpers. The owner hospital holds the spending records of the patients and publishes that data to the blockchain. The helper hospitals help the insurance company to query for the patient's spending data on the blockchain and perform homomorphic computations on the results. However, the helpers cannot learn anything about the patient's spending data, as long as there is no collusion between helpers. We deploy our system on the Ethereum blockchain and give a final performance evaluation.

Dedication

I would like to dedicate my work to my parents who have always loved me unconditionally. This thesis is also dedicated to my wife and children who always motivate me to succeed and supported my every decision.

Acknowledgements

I would like to express my sincerest gratitude to my professor, Dr. Rongxing Lu for his patient, guidance and encouragement throughout my time as his student. I have been very lucky to have a supervisor who cared so much about my work. I would also like to thank all the members of staff at the University of New Brunswick who have been there to help me with every aspect of academic and student life.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Almost every aspect of modern life has been disrupted and improved by information technology, and the blockchain technology is one of the fastest growing sections of IT at the moment. Just like the Internet, this new technology will not only make doing business easier, cheaper and more efficient, but also increase the security of current methods of business. The most popular usage of blockchains is to create and trade cryptocurrencies, so the majority of blockchain data is cryptocurrentcy transactions, but the technology is not only restricted to this use.

In our research, we will focus on the public blockchain technology. In that regard, the blockchain is a public, trusted and shared ledger maintained by all of its participants. The network is peer-to-peer with no central authority, and is governed by predefined rules, enforced by the collaboration of participants.

It is public because all participants have public access to the same version of the ledger. The stored information is highly persistent and tamper-resistant, making the blockchain a trusted ledger. These characteristics give blockchain the potential to revolutionize data storage and its business applications. In this thesis, we propose a blockchain-based medical insurance storage system which can be used for storing and retrieving medical insurance information in a secure and privacy-preserving way.

1.1 Background of blockchain systems

In this section, we will take a brief look at some blockchain platforms - Bitcoin, Ethereum and Hyperledger Fabric.

1.1.1 Bitcoin

In 2008, Satoshi Nakamoto published a paper titled "Bitcoin: A Peer-to-Peer Electronic Cash System" [50] describing a decentralized electronic cash system [2]. The system has no central authority to issue currency and to validate transactions. Nakamoto introduced the Proof-of-Work algorithm, which synchronizes the state of the decentralized system every 10 minutes by holding a global election. Using this innovation, the system can prevent double-spending, where a digital currency unit can be spent twice. This is a significant improvement from previous digital currencies that tried to solve the double-spending problem through a central clearinghouse.

The bitcoin network came into existence on January 3, 2009 when Satoshi Nakamoto mined the first block, reaping a reward of 50 bitcoins. The network has since then been revised many times by other programmers. The true identity of Nakamoto remains unknown even after he left the bitcoin scene in 2011 and handed over the network development to a group of volunteers. The value of bitcoin increased from almost nothing in 2009 to an all-time high of $19,783.06 in 2017, and dropped to around $4000 in February 2019, bringing the total market value of the bitcoin network to around $70.5 billion at the time of this writing. The most notable thing is that there is no individual or group that has control over this network, which is open source and operates on fully transparent mathematical principles and consensus among participants.

1.1.2 Ethereum

After the birth of Bitcoin, people began to realize the power of blockchain and tried to apply it to other applications other than cryptocurrencies. The choice included either to build upon the Bitcoin network or start a new blockchain. Building on Bitcoin has the advantage of the available computing power of the current participants, but the applications would be limited by the internal constraints of the network. The smart contract implementation of Bitcoin is

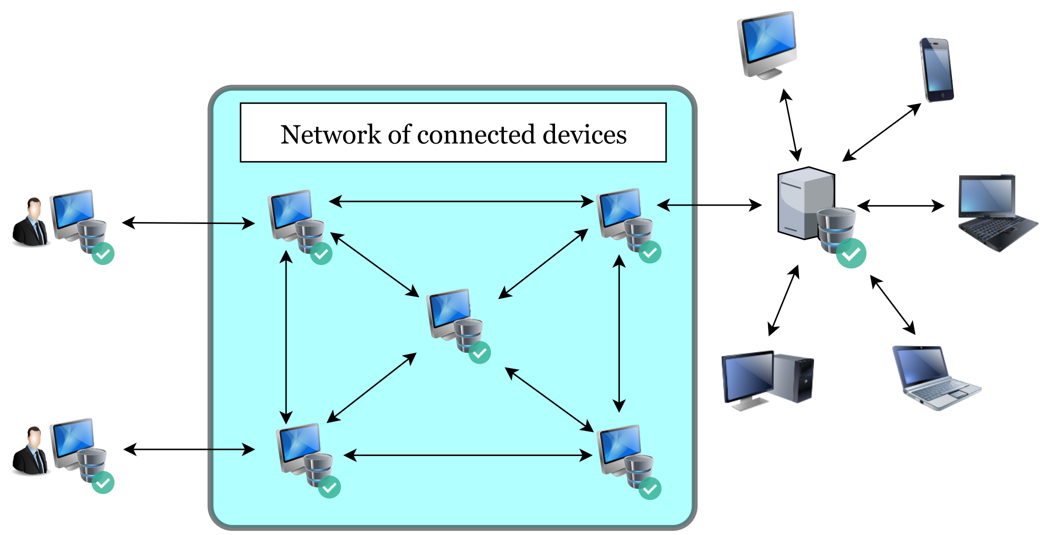

Figure 1.1: A distributed ledger system

very restricted, as well as the transaction types, data types and size of data storage, which severely limits the sorts of applications that can run directly on the blockchain.

In 2013, Vitalik Buterin invented the Ethereum blockchain [18] to do things Bitcoin couldn't. Ethereum was conceived with the idea of a general purpose blockchain platform, equipped with a powerful Turing-complete language capable of writing complex smart contracts for any types of application. By using such an approach, the mechanisms of peer-to-peer networks, blockchain and consensus are abstracted. Developers can focus on their applications design and development without worrying about these details.

1.1.3 Hyperledger Fabric

Hyperledger was started in 2015 by the Linux Foundation as an open source blockchain platform with to support blockchain-based distributed ledgers. It focuses on developing blockchains with the goal of improving performance and reliability of global business transactions from major technological, financial, and supply chain companies. The project aims to enable multiple individual teams to collaborate in order to develop open protocols and standards, by providing a modular framework that supports different components for different uses. This approach may support a variety of blockchains, each with its own consensus, storage models, services for identity, access control, and contracts.

The Hyperledger project consists of many individual blockchains contributed by different companies, and Hyperledger Fabric is one of them. It is the work of IBM and Digital Asset. Like Bitcoin and Ethereum, it has a ledger, uses smart contracts, and is a system by which participants manage their transactions. But this is where the similarities end. Hyperledger Fabric differs from public blockchain systems in that it is private and permissioned [1]. While open permissionless systems allows unknown identities to participate in the network and utilize protocols like “Proof-of-Work” to validate transactions and secure the network, the members of a Hyperledger Fabric network enroll through a trusted Membership Service Provider (MSP). Peers in Hyperledger Fabric can be divided in to several types: *Committer*, *Endorser* and *Orderer*.

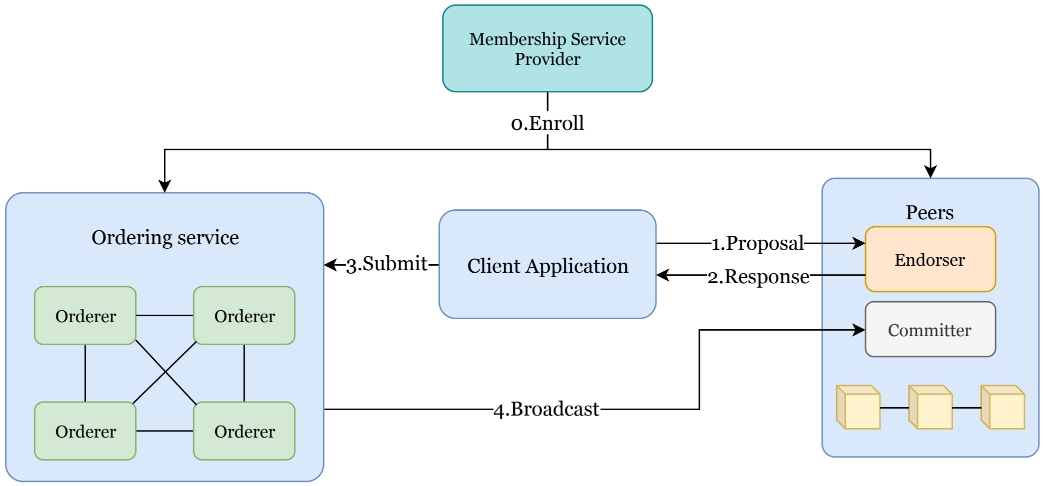

Figure 1.2: Hyperledger Fabric Architecture

Every peer is a *committer*; *endorsers* validate transactions; *orderers* collect transactions into blocks and broadcast them to other peers. Figure 1.2 shows the high-level architecture of a Hyperledger Fabric system.

The components of Hyperledger Fabric are pluggable and configurable. Most notably, consensus protocols can be customized to suit specific use cases and trust models. MSPs can be configured with their own notions of identity and the rules to govern those identities. Lastly, ledger data can be stored in multiple formats.

What if some participants want to have transactions between themselves that are not known to others, for example, when they are offering a special price to a selected group of participants? Hyperledger Fabric also allows its participants to create channels, enabling them to have private ledgers of transactions. This is an especially important option for networks where some participants might be competitors and don't want every transaction they make known to every participant. If two participants form a channel, then only those participants have copies of the ledger for that channel.

1.2 Advantages and Challenges of Blockchain

Blockchain technology seems like a great solution to increase the security and integrity of our databases, but it is not without its weaknesses. While blockchain can provide decentralization, transparency, immutability and availability to your data, public blockchain networks such as Bitcoin and Ethereum suffer from privacy-leaking and low-performance issues.

1.2.1 Advantages

1.2.1.1 Decentralization

The most notable advantage of blockchain is eliminating the need for a trusted third party. Everyone can verify the validity and authorization of blockchain transactions. Traditional databases need to be hosted and administered by an entity, and no matter how trusted that entity is, there is still the chance that somebody who has access to the system can alter the data. The human factor is always the weakest link in a security system. By removing the third party from the picture, not only does it increase the security of the system, it also reduces the cost for organizations and greatly simplifies processes.

1.2.1.2 Transparency

Blockchain is a distributed ledger, that means every participant in the network have access to an identical copy of the database, so the system is transparent, and no modification of the ledger can happen without everyone knowing about it. This is important in scenarios such as digital health care recording as patients want to see their data and what happens to it.

1.2.1.3 Immutability

It is very difficult to modify some data after it is written into the blockchain, and the longer a transaction has been present in the blockchain, the harder it gets to modify it. In order to change a block, the attacker needs to recalculate every subsequent blocks, which is not possible without the collusion of the majority of nodes in the system.

1.2.1.4 Availability

Because the ledger is replicated and updated in every node in the network, there is no single point of failure. As the number of node grows, the availability just becomes higher and higher, and if some nodes becomes unavailable, the network still functions normally.

1.2.2 Challenges

1.2.2.1 Privacy

Because the blockchain is fully transparent, everyone can know what everyone else is doing, and even though parties are anonymous on the blockchain, confidential information can still be leaked. An attacker can easily trace an user's transactions and obtain sensitive information. This is especially important in the financial sector, where users usually do not want their financial situation to be known by everybody. In this aspect, a system with a central authority may have the advantage where they have access control to the data in one place, thus make it easier to hide information.

1.2.2.2 Performance

Performance has always been a problem with blockchain. The two most popular cryptocurrencies - Bitcoin and Ethereum - support less than 20 transactions per second, a very small number compared to those supported by modern transactional database systems. While the Proof-of-Work consensus system helps ensuring data immutability, it also places a burden on the network. The mining process requires a huge amount of computing power and the larger the network grows, the more power it takes to generate a single block.

1.2.3 Current Research Focus

In this section, we will take a look at current research focuses in blockchain. A summary is shown in Table 1.1.

1.2.3.1 Security

A significant portion of current research on blockchain is focused on improving the security and privacy of blockchain networks. With the public's growing interest in Bitcoin as a method of payment, security incidents of the Bitcoin network also result in increased economic losses for its user [62]. There are several identified vulnerabilities, including computation power-based attacks (51% attack, selfish mining attack)[9], transaction data malleability problems [21], and deanonymization by transaction linking [42]. Many of the proposed solutions proposed by researchers are just brief suggestions whose effectiveness have not been concretely evaluated.

1.2.3.2 Scalability

As mentioned in previous work [62], current solutions for public blockchain have limited scalability and high latency. For example, Bitcoin can only process an average of 7 transations per second, while the Ethereum network can process 20 transactions per second. The numbers are clearly too small to serve an ever-increasing user base of cryptocurrencies. To address this limi-

Blockchain Research	
Security	Power-based attacks [9]
	Transaction Data Malleability [21]
	Authentication [6]
Privacy	Anonymous Ownership [48]
	Transaction Linking [42]
	Transaction Mixing [31]
Scalability	Side-Chain Transactions [5]
	Off-Chain Transactions [53, 24]
	Consensus [39]
Smart Contract	Programming Language [18]
	Security Issues [60]
	Protocols [11]

Table 1.1: Current Blockchain Research Focuses

tation, many solutions have been proposed, such as using side-chains [5], increasing the block size, using other consensus protocols [39], or using off-chain channels [53, 24]. Side-chains are parallel, separate blockchains designed to handle transactions at higher speed and lower cost. They are often used between parties who make lots of transactions with each other. Essentially, a side-chain aggregates a large number of transactions into one transaction to be processed on the main chain at a later time to reduce cost. Similarly, an off-chain solution is also a way to aggregate transactions, but without using a blockchain. Users can use a centralized server to make transactions, and combine them all into one final transaction to commit on the main chain. Finally, some blockchain networks are tending towards the use of "Proof-of-Stake"(PoS) consensus mechanism. Unlike "Proof-of-Work"(PoW), where miners compete to solve a problem in the least amount of time, the creator of a block in PoS is chosen in a deterministic way depending on its wealth - also known as ‘stake’. It can easily be seen that the cost for PoS consensus is only a fraction of PoW, due to the fact that there is no competition. On the other hand, because PoS based systems may prove cheaper to attack than PoW, new security protocols for PoS also need to be developed, such as Ethereum’s Casper Protocol [19].

1.2.3.3 Privacy

Studies have shown evidence that the Bitcoin network does not achieve complete anonymity. Meiklejohn and Orlandi [48] presented a definitional framework for virtual currencies that utilizes privacy-enhancing techniques to increase the anonymity of coin ownership. Koshy et al. [42] analyzed a Bitcoin traffic pattern and came to the conclusion that it is possible to map some subset of Bitcoin addresses to an IP address by simply observing the transaction relay traffic. Feld et al. [26] created a framework that traverses the Bitcoin network to generate some statistics regarding its size and distribution among autonomous systems (AS). They figured out that an average peer-list contains addresses that mostly reside in the peer’s own AS. From this information, the authors claim that transaction linking could be possible.

1.2.3.4 Smart Contracts

A smart contract is a computer code platform that contains a set of rules under which participants to that contract agree to interact between themselves. Smart contracts are deployed in a decentralized environment - such as on top of a blockchain platform - where contract terms are executed by the system when the terms are fulfilled. Based on the Bitcoin protocols, Bigi et al. [11] introduced a decentralized smart contract protocol inspired by BitHalo [12] and validated its feasibility. Using a combination of game theory and formal models, the authors claimed that a decentralized smart contract system can be a promising approach and worthy of being studied and developed further. Wan et al. [60] used the Bitcoin network as a way of providing a time-stamping service and proposed an electronic signing protocol between two parties with that service. Furthermore, one can use smart contracts in various environments and industries depending on the purposes. For example, Kishigami et al. [40] developed a prototype of a Blockchain-based digital content distribution system. The authors presented the idea to about one hundred people including creators, content owners and digital content stake holders. The impressive and attractive point was the decentralized mechanism which can be used for Digital Rights Management.

Blockchain Applications	
Finance	Banking [52]
	Risk Management [49]
	Peer-to-peer Financial Market [29]
	Blockchain-as-a-service [4]
Internet of Things	Secure Data Access [43]
	Privacy-Preserving Data Sharing [20]
Healthcare	Universal Medical Record Storage [47, 3]
	Privacy-Preserving Medical Record Data Sharing [47]
Supply Chain	Inventory Management [38]
	Product Tracking [38]
	Product Verification [38]
	Order Recording [38]
	Product Information Sharing [38]

Table 1.2: Blockchain Applications [65]

1.3 Blockchain Applications

In this section we will take a look at potential applications of blockchain in Finance, Internet of Things, Healthcare and Supply Chain Management. A summary is shown in Table 1.2.

1.3.1 Finance

The appearance of public blockchain systems such as Ethereum [18] and permissioned systems such as Hyperledger [1] sparked public's interest in revolutionizing the traditional financial and business services [65].

Blockchain can potentially disrupt many areas, including banking [52], risk management [49], enterprise transformation or peer-to-peer financial market

[29]. Since 2016, large software companies like IBM [34] and Microsoft [4] also began to offer Blockchain-as-a-Service .

1.3.2 Internet of Things

Internet of things, one of the most promising information and communication technologies, has been a hot topic for research recently [37]. However, the security of IoT devices leaves a lot to be desired, particularly in anonymity and authentication. Security and privacy are the two major concerns in computer systems that involve IoT devices. Blockchain-based IoT provides security and privacy - two major concerns in IoT-related computer systems. Rifi et al. [43] proposed a fully decentralized blockchain-based architecture that uses smart contracts to implement secure data access in IoT. Hardjono and Smith [20] proposed a privacy-preserving mechanism to share sensor-data securely and reliably.

1.3.3 Healthcare

Blockchain can potentially create a huge revolution in the healthcare ecosystem by bringing security and privacy to patient healthcare management [47, 3]. Individuals will be responsible for handling their own records, and getting the overall control of their own data with the support of this technology. By using a shared ledger, the system would be accessible by all the entities involved in the process no matter which electronic medical system they use. Not only does this offer higher security and transparency, it also allows doctors to spend more time on patient care and treatment. Moreover, statistics of researches will also become easier to share, which, in turn, would facilitate clinical trials and treatment therapies for rare diseases.

In a healthcare system, smooth data sharing between healthcare solution providers are so important that it can lead to improved accuracy in diagnosis, effective treatments, and cost-effective ecosystem. As patient data grows more and more everyday, proper utilization of resources is required to effectively gather and take advantage of insights from that data.

Blockchain for healthcare makes uses of a commonly distributed ledger to allow multiple entities of the healthcare ecosystem to stay in sync and share data. With such a system in place, the participants can share and keep track of their own data and other activities happening in the system without having to worry about security and integrity.

1.3.4 Supply Chain Management

Because blockchain technology allows us to securely and transparently track all types of transactions, it can provide a huge improvement in the supply chain. Every time a product changes hands, the transaction could be documented, creating a permanent history of a product, from manufacture to sale. This could dramatically reduce time delays, added costs, and human errors. Specifically, blockchain can improve the following tasks [38]:

- Recording the quantity and transfer of assets - like pallets, trailers, containers, etc. - as they move between supply chain nodes.
- Tracking purchase orders, change orders, receipts, shipment notifications, or other trade-related documents.
- Assigning or verifying certifications or certain properties of physical products; for example determining if a food product is organic or fair trade.
- Linking physical goods to serial numbers, bar codes, digital tags like RFID, etc.
- Sharing information about manufacturing process, assembly, delivery, and maintenance of products with suppliers and vendors.

1.4 Our Contribution in This Thesis

This work is focused on the application of blockchain on a medical insurance storage system. In an ideal and basic medical insurance business, there are a group of hospitals, a patient and an insurance company. The insurance company can know a sum of the patient's specified spending records, however the company cannot learn the details of the spending records. The other hospitals can help the insurance company to process a patient's spending records without learning anything about the records. Specifically, wee propose a privacy-preserving system which utilizes blockchain technology to store and process medical insurance data, which is characterized by the following features:

- Our system is fully decentralized, as there is no trusted third party to provide authentication.
- The data is verified and securely stored before being included in the blockchain, which provides high credibility to users.
- We adopt the $(2, 3)$ threshold secret sharing protocol among participating hospitals, so that the final data can only be obtained if at least 2 hospitals respond to the insurance company's request.
- Every data stored on the blockchain can be efficiently verified by anyone, and even though the encryption is secure, the verification requires very little computational power. Results are obtained with the use of efficient homomorphic computation.

1.5 Thesis Outline

The rest of the thesis is organized as follows:

- Chapter 2: Technical Background of Blockchain, Security of Blockchain and Related Work about Blockchain applications in Healthcare.
- Chapter 3: Background of the cryptographic techniques employed to build our system.
- Chapter 4: Our System Model, Security Model and Design Goal.
- Chapter 5: Our proposed System Design.
- Chapter 6: Security analysis and performance evaluation of the system.
- Chapter 7: Summarizes the conclusions and contributions of this thesis and proposes our future research work.

Chapter 2

Technical Background of Blockchain

The blockchain can be thought of as a distributed ledger. The ledger is typically shared across a large network containing many computers and it is completely public. Every time the network makes an update to the ledger, it is automatically updated and downloaded to every computer on the network. Blockchain technology is secured with cryptographic techniques, making it near impossible for hackers to make changes to past records. The only way to change history would be to possess more computing power than the rest of the network combined, which is why it is more secure to have more nodes/-computers running the blockchain.

2.1 Blockchain Technical Background

In this section, we will briefly take a look at blockchain building blocks: *Transactions*, *Blocks*, *Mining*, *Consensus* and *Smart Contract*.

2.1.1 Transactions

Transaction is the basic operational unit in Blockchain, which represents the act of transferring a value from a sender's address to a receiver's address. From the public key, the address can be derived by first hashing it with SHA-256 and then with RIPEMD-160 [22], then prepend the result with a version number and a checksum for error detection [57]. One can use a new key and address for each transaction to prevent comparison-based attacks on signatures and coins flow tracking. To encode transactions, Bitcoin uses a simple stack-based Turing-incomplete language called Script, while Ethereum uses a Turing-complete language named Solidity. In this research, we will take a look at the high level architecture of Blockchain, but not delve too deeply into the technical details, because our goal is to design a system that operates on top of existing Blockchain platforms (Ethereum or Hyperledger). The most significant breakthrough in designing Blockchain transaction is solving the double spending problem in an ingenious way. The critical components of a transaction are a hash value as the transaction identifier and a list of inputs and outputs. An input can only be used once. In other words, if a transaction refer to an input that has already been used, that input will be considered “double-spent” and the the transaction will be rejected by the network.

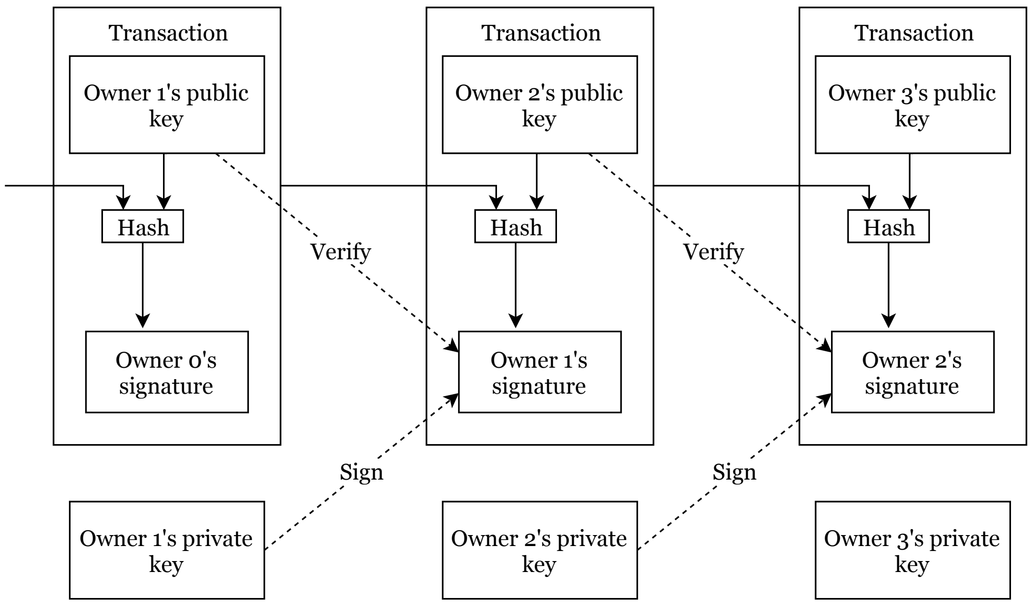

Figure 2.1: Structure of chaining blockchain transactions [50]

2.1.2 Blocks

At its core, blockchain is a chain of blocks where each block is linked to its previous block by reference to the previous block header’s hash [46]. We will take a look at the block structure that is present in the Bitcoin Blockchain. A block contains three parts: block header, transaction counter and transactions. Transaction counter is the total number of transactions in this block while transactions is the list of all transactions. The block header is a complex structure including block version, parent block hash, Merkle root, timestamp, difficulty target (nBits) and nonce. The block validation rules are dictated by block version. *Previous block hash* is a double SHA256 hash of previous block's header. *Merkle root* is a double SHA256 hash of the Merkle tree of all transactions. *Timestamp* contains the approximate creation time of the block in the Unix epoch time format [8]. *Difficulty target* represents the difficulty of the Proof-of-work puzzle in order to generate a new block, and *nonce* is the answer to the puzzle. The block structure is shown in Figure 2.2. There is a special block called *Genesis Block*, which is the first block in the Bitcoin blockchain and has no previous block hash. This block structure makes it easy to verify transactions because the Merkle root can reflect the complete list of transactions in this block. Any attemps to tamper with the transactions would result in a different value of Merkle root and would be detected. On average, new blocks are added to the Blockchain every 10 minutes by the process of mining which is explained below.

2.1.3 Mining

Mining is the core mechanism of blockchain technology and it includes the following tasks: transaction validation, block validation, creating a new block, performing the Proof-of-Work algorithm and fetching reward [46]. In the process of mining, miners are required to solve a mathematical puzzle which requires a lot of computing power. Successfully solving the puzzle enables the

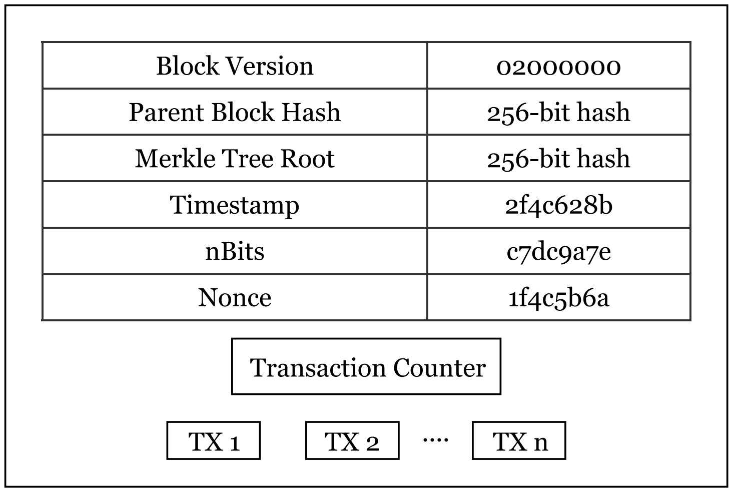

Figure 2.2: Block Structure

miner to create a new block of transactions and commit it to the blockchain, then the miner can claim newly generated coins as the reward. The mathematical problem in each block is extremely difficult to solve, but it is very easy to verify whether a solution is correct. There are multiple valid solutions for any given block - only one of which needs to be found for the block to be solved. The mining process is depicted in Figure 2.3.

The network automatically adjusts the difficulty of the mathematical problem so that a block can be mined every 10 minutes. After every 2016 blocks (solved in about two weeks), all nodes compare this goal with the actual mining speed of the network, and adjust the difficulty target by the percentage difference. The network comes to a consensus and automatically increases (or decreases) the difficulty for subsequent blocks.

As mentioned above, the blockchain is a chain formed by connecting every

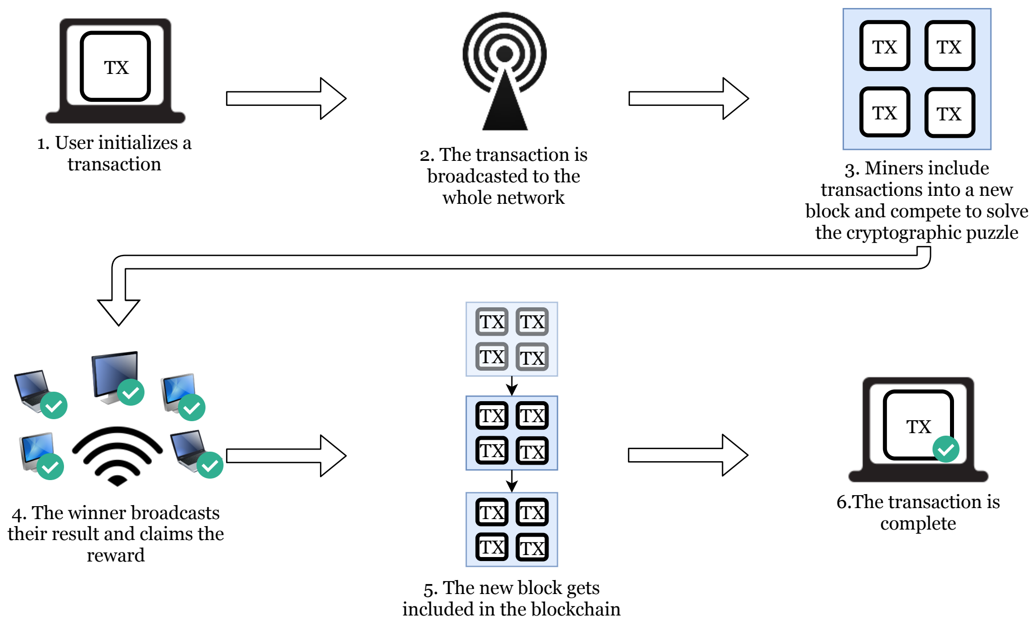

Figure 2.3: The mining process for a new block

block with its previous block. It's also possible for the chain to have temporary splits, for example, when two miners arrive at two different solutions for the same block at the same time. By design, the blockchain can resolve these splits in a short time, so that only one branch of the chain survives.

2.1.4 Consensus

In peer-to-peer networks, the consensus is a concept describing the distributed coordination of nodes. In blockchain, it is a means of coordinating all nodes in a blockchain network to agree to a single version of truth.

In recent years, the consensus algorithm in distributed systems has become an active area or research involving proof-based algorithms (Proof-of-Work [50], Proof-of- Stake [58] and etc.) and Byzantine Fault Tolerance-based algorithms which employ the voting system to solve consensus problems. However, among these algorithms, only Proof-of-Work (PoW) has proven to be adequately resistant to Sybil attacks [8]. In blockchain, the PoW algorithm is used for block validation within the process of mining. Sometimes, forks may occur when two nodes simultaneously announce a valid block or the Bitcoin protocol has updated. The first case is a natural fork and is an undesirable situation in which the blockchain employs PoW to resolve the forking problem. According to [50], mining is continued on the 'longest' fork which involves the highest volume of computing power. In other words, only the most extended chain can be chosen as a result of the PoW consensus algorithm which is also called the main chain. The other divergent chains are called orphaned chains which will be discarded. Nevertheless, this situation has a potential threat: if the attacker(s) amasses more than 50% computing power of the whole network, they can rewrite the blockchain and change the transactions as they wish. This is called the 51% attack and it has been rigorously studied by researchers, leading to innovative ways to prevent the attack. In the second case, the forks that result from version updating can be resolved by upgrading the version of blockchain. The Proof-of-Work consensus mechanism in blockchain is shown in Figure 2.4.

The alternative consensus mechanism to Proof-of-Work is called Proof-of-

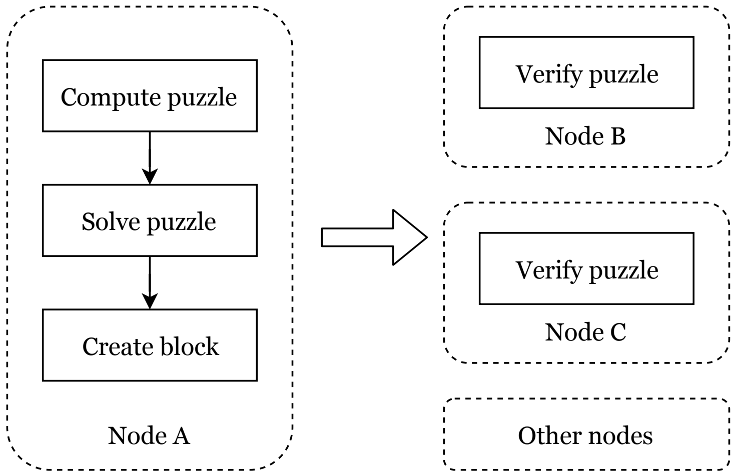

Figure 2.4: The PoW consensus mechanism [44]

Stake (PoS) [58]. Unlike PoW where anyone can be a miner and create blocks, participants in the PoS consensus-building process are required to have a legitimate stake in the blockchain - for example, those who actually own Bitcoins in the Bitcoin network. Instead of having participants solve a cryptographic puzzle, this algorithm selects an individual to validate new blocks of transactions based on their proportion of stake in the network. The larger an individual's share of the stake is, the higher their chance is to get selected. Of course, PoS is also not perfect. By favouring those who have heavily invested in the network, as time goes on, the network gets more and more centralized. This is undesirable for a decentralized network that aims to be truly robust.

2.1.5 Smart Contract

Smart contracts is not a new concept, but it only became a hotspot of research with the appearance of Blockchain. The terminology of smart contracts was first theorized by Szabo [56] in the early 1990s, but researchers only began to realize the potential and benefits of them since the Bitcoin came into existence in 2009. The most attractive part in smart contracts lies in the scenario that it is not necessary for nodes to trust each other and there is no need for a trusted intermediary when transferring values between each other in a trustless peer-to-peer network. Until now, there is no universally agreed upon definition of smart contracts. According to [8], smart contracts can be defined as a secure and unstoppable computer program representing an agreement that is automatically executable and enforceable. From this point of view, smart contracts are computer programs and are interpreted as code. Like contracts, they encompass the agreement between two parties. Smart contracts will be executed automatically when some conditions are met. They are enforceable, which means that they should be executed by measurements of controls without any mediation. In Blockchain, real smart contracts should be only carried out with code. In other words, the code is law [8]. Another critical issue in smart contracts is how to carefully design the contracts to make them secure and unstoppable. The smart contracts should be fault tolerant and they should be executed in reasonable amounts of time [8]. Kosba et al. [41] have explained the future use of smart contracts in Blockchain and showed full of expectation to the promising future in this field.

2.2 Security of blockchain

The core principle in blockchain is decentralization and it has important implications for security. A centralized model, such as a traditional bank or payment network, depends on access control and vetting to keep bad actors out of the system. By comparison, a decentralized system such as blockchain pushes the responsibility and control to the users. By basing the network security on Proof-of-Work, the network can be open without requiring blockchain traffic to be encrypted [2].

On a traditional payment network, for example, a credit card system, the payment contains the user's credit card number - the private identifier - so it is open-ended. Anyone with access to the identifier can 'pull' funds and charge the owner again after the initial charge. Thus, the payment network has to be secured end-to-end with encryption and must ensure that no eavesdroppers or intermediaries can compromise the payment traffic, in transit or when it is stored [2]. If a malicious user gains system access, current transactions can be compromised and new transactions can be created with the payment tokens. Even worse, compromised user data may lead to identity theft and customers themselves need to take action to protect their accounts.

It is drastically different with blockchain. A blockchain transaction only authorizes a specific value to a specific recipient and cannot be forged or modified. It does not expose any private information, such as the identities of the parties, and additional payments cannot be authorized using old information. Therefore, a blockchain payment network does not need to be encrypted or protected from eavesdropping. In fact, blockchain transactions can be transmitted over an open public channel, such as unsecured WiFi or Bluetooth, with no loss of security.

Blockchain's security model puts a lot of power in the hands of the users, along with the responsibility for maintaining the secrecy of the keys. For most users that is not an easy task, especially those using general-purpose computing devices such as smartphones or laptops. Although blockchain's decentralized model prevents the type of mass compromise seen with credit cards, many users are not able to properly secure their keys and get hacked, one by one.

2.2.1 51% attack

The 51% is a computing power-based attack on a Proof-of-Work blockchain such as Bitcoin, by an attacker possessing more than 50% of the network's computing power. Normally, a block cannot be altered once it is finalized, since modifying a block would mean modifying all subsequent blocks, as it is almost impossible for an entity to create blocks fast enough to catch up with the network. However, an attacker or group of attackers controlling the majority of the network's computing power can do just that. With more computing power than all the other users combined, they can create new blocks faster on average, theoretically allowing them to monopolize the mining of new blocks and earn all of the rewards. Or worse, they can execute the double-spending attack by sending a transaction, then reverse it, making it appear as though they still had the coin they just spent. Double-spending is the digital equivalent of a currency counterfeit and the basic cryptographic hurdle the blockchain was built to overcome, so a network that allowed for double-spending would quickly suffer a loss of credibility.

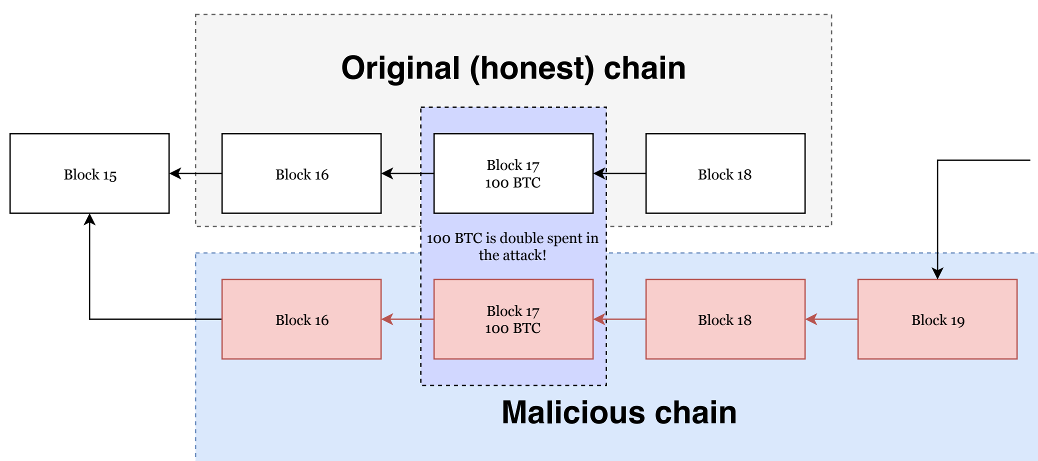

Figure 2.5: A diagram of the double-spending 51% attack. The 100 bitcoins spent in Block 17 of the original chain is used again in the malicious chain, which later outgrows the original to become the main blockchain.

It should be noted that even by controlling 51% of the network, it is still highly difficult to modify the blocks and transactions locked in earlier to the start of the attack. The deeper a transaction is buried in the blockchain, the more difficult it would be to change it. Some bitcoin clients implement 'checkpoints', where some block headers are hard-coded directly into the software, to prevent history from being rewritten past a certain point in time. On the other hand, it is still possible to perform a form of 51% attack with less than 50% of the network's mining power, but with a lower probability of success. Smaller blockchain networks are more likely to suffer from 51% attacks than large networks such as Bitcoin or Ethereum.

- **Ghash.io**. In July 2014, the mining pool ghash.io surpassed 50% of the bitcoin network's computing power for a short time, leading the pool to voluntarily commit to reducing its share of the network. The pool later released a statement that it would not overcome 39.99% of the total mining power in the future [28].

- **Krypton and Shift**. Krypton and Shift, two small Ethereum-based blockchains suffered 51% attacks in August 2016 [16]. After the attack, Krypton switched over to the Proof-of-Stake consensus to prevent such an event from happening again in the future.

- **Bitcoin Gold**. In May of 2018, Bitcoin Gold, the 26th-largest cryptocurrency at the time, suffered a 51% attack [27]. The attackers controlled a vast amount of Bitcoin Gold's hash power and performed double spending attacks by trading the same coins for other cryptocurrencies over and over again. The attack went on for several days even with Bitcoin Gold repeatedly attempting to raise the exchange threshold, eventually stealing more than $18 million worth of Bitcoin Gold.

- **Ethereum Classic**. In January of 2019, Ethereum Classic - one of the top 20 cryptocurrencies at the time - was hit by a 51% attack and $1.1 million worth of ETC has been double spent. The network developers have since then created a monitoring alert systems to detect attacks faster.

2.2.2 Denial-of-Service attack

The distributed nature of blockchain network and its consensus protocol cause Denial-of-Service(DoS) attacks to have no little adverse effect on network functionalities, hence attackers have to launch a powerful Distributed-Denial-of-Service(DDoS) to disturb the networking tasks. Unlike DoS attack where a single attacker carries out the attack, in DDoS, multiple attackers launch the attack simultaneously. DDoS attacks are both cheap to carry out and disruptive in nature. Malicious miners having access to a Botnet can perform a DDoS on competing miners, effectively taking the honest miners out of the network and increasing the malicious miners' hashrate. In these attacks, the network resources of honest miners are exhausted, disrupting their access to genuine users. For example, an adversary can control a large number of clients and send lots of fake transactions to a honest miner, congesting that miner with requests. After some time, the miner will be likely to start discarding all incoming requests including those from honest clients. In [59], authors provide a extensive empirical analysis of DDoS attacks in the Bitcoin network by documenting the following main facts: There had been 142 unique DDoS attacks on 40 Bitcoin services and 7% of all known operators were victims of these attacks. Attackers are more likely to target exchange services and large mining pools because a successful attack on these will enable the adversary to gain huge profit as compared to attacking an individual or small mining pools.

2.2.3 Sybil attack

Sybil attack is an attack where a majority of nodes on the network are taken over and controlled by malicious actor(s). Any decentralized and permissionless peer to peer network must guard against this form of attack, including the likes of BitTorrent as well as Blockchain. If the network is filled with clients that an attacker controls, a honest user would then be very likely to connect only to attacker nodes. Even though the node count does not have any use in the Bitcoin network, completely isolating a node from the honest network can be helpful in the execution of other attacks. If you are under a Sybil attack, you can be exploited in the following ways:

- The attacker can isolate you from the network by refusing to relay blocks and transactions from everyone.

- The attacker can relay only blocks that they create, effectively putting you on a fork and then also leaving you open to double-spending attacks.
- If you rely on transactions with 0 confirmations, the attacker can just filter out certain transactions to execute double-spending attacks.

2.2.4 Eclipse attack

Rather than attack the whole network like in a Sybil Attack, an Eclipse Attack is a means of attacking a decentralized network through which an attacker seeks to isolate and attack a specific user. A successful Eclipse Attack an adversary to isolate and subsequently prevent their target from obtaining a true picture of the real network activity and the current ledger state. The Eclipse attack is shown in Figure 2.6.

This attack is made possible because in a decentralized network, every node doesn't simultaneously connect to every other nodes on the network. Instead, for efficiency, a node only connects to a select group of other nodes, who in turn are connected to another group of their own. The goal of the attacker is to hijack all of these connections. Generally, an attacker would have to control a Botnet of host nodes with their own IP address, and work out the neighboring nodes of an intended victim. Of course the effort required to achieve this varies by the construct, size and nature of a network. After the

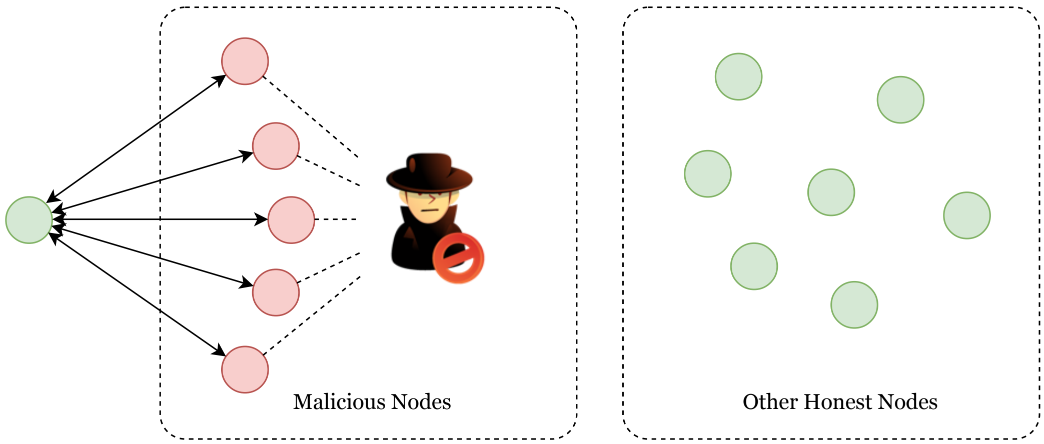

Figure 2.6: A diagram of the eclipse attack. The honest node is essentially isolated from the rest of the network and can only communicate with malicious nodes.

victim node logs off and then rejoins the network the next time, all of the victim's connections may have been nodes controlled by the attacker.

Once a malicious actor has taken control of all outgoing connections of a user, they are able to exploit them by, for example, carrying out a 0 confirmation double spend attack. In a scenario where User A is the malicious actor, User B is the isolated node and User C is another honest network entity, then User A would be able to send a payment to User C and send the same transaction to User B. Because all of User B's connections are controlled by User A, User B is unaware that those funds have already been spent as the information they receive is manipulated and suppressed. After User B accepts the coins and later connect to the 'true' blockchain, they will find out that they have been tricked and have in reality received nothing.

An eclipse attack may also be executed to attack the blockchain itself, hijacking the mining power of an isolated node(s). The victim, only seeing the ledger that the attacker is showing, will lend their support to this variant of the chain. If the attacker can attack enough users - some of which may control significant amounts of hashing power - they would be able to present their own chain as a legitimate fork to the 'true' ledger, then try to gain enough support and make that the main ledger.

As noted in [32], other potential Eclipse Attack effects on a PoW network such as Bitcoin include:

- Engineering block races: When two miners mined their blocks at the same time, one block would be included in the chain while the other will be discarded. An Eclipse attack would mean the attacker can 'hide' the discovery of mined blocks from victims and then release once a competing block has been found, wasting the eclipsed miner's mining effort on orphan blocks.

- Splitting mining power: A malicious actor could try to eliminate eclipsed miners from the network and in doing so reduce the amount of active miners, effectively making it easier to launch 51% attacks.

- N-confirmation double spend: To guard against double spend attacks, most merchants wait until the payment transaction is included in a block and buried under a number (N) of blocks before providing their service. In spite of this, if the merchant and enough miners have been eclipsed then an attacker can still launch a double spend attack. The attacker shows their transaction to the eclipsed miners, who include the transaction into new blocks that get added to their ledger. The eclipsed miner can only see this eclipsed ledger and assume that the transaction is confirmed, and provide service to the attacker. Afterwards, when the miners and merchants are able to connect to the real network, their ledger will be discarded, leaving the merchant with nothing.

2.2.5 Selfish Mining

The selfish mining attack is a way for mining pools to increase their returns by not playing fair, it is an attack on the integrity of a blockchain network. In a selfish mining attack, the attacker - who possesses a large portion of the network's hashing power - keeps on mining new blocks without releasing them to the network until they're about to be caught up by other miners. By working on new blocks internally for as long as they can, they actually increase the block discovering speed, which means generating higher revenue [25]. But the late release of blocks results in the chain getting forked and a large number of blocks discarded, not only wasting the effort of other miners but also undermining the integrity of the network.

Selfish mining this can be seen in some cryptocurrencies where pool shares are not so fairly distributed, though it is more difficult to carry off with Bitcoin. A mining pool may find it advantageous to increase in size to be able to perform selfish mining. With over 51% of network power, the returns earned by a selfish mining pool can be justified as probabilistically, they have an advantage in terms of hashing the next block.

Contributors have advocated several BIPs - or Bitcoin Improvement Proposals - to lower the probability of a selfish mining attack. One solution is to randomly assign miners to various branches when a fork occurs - or to provide a threshold limit to which a mining pool can reach. This is the same as Governments trying encourage competition by halting natural monopolies. Another solution is to take into account the timestamp when a block was released. For example, if a miner releases a long list of blocks at once, the rest of the network would weight their validity against the timestamp they were hashed and the timestamp they were reported to the network.

2.3 Related Work

Researchers have realized blockchain's potential to enable better health data sharing and ownership. Using a public or private blockchain to actually store clinical data is one example - for example, Yue et al. [63] described a "Healthcare Data Gateway" (HDG) which utilizes private blockchains to enable patients to manage their own health data. Similarly, Ivan [35] described an implementation of blockchain-based Personal Health Record, where healthcare data is encrypted but stored on a public blockchain. Another example is MedChain [47], where a permissioned network of medication stakeholders (including the patient) could be used to facilitate medication-specific data sharing between patients, hospitals, and pharmacies. While we assume that the idea of storing actual clinical data on a blockchain - permissioned or public - would have substantial privacy and scalability concerns, it is important to continue to understand the privacy and security implications of on-chain data storage [30].

On the other hand, we can leverage blockchain not for the storage of the actual clinical data, but for facilitating management or governance of that data. Zyskind et al. [67] have described a general-purpose decentralized access and control manager for encrypted off-chain data, where the data is stored off chain, but access control policies are enforced in the blockchain layer. In the healthcare space, FHIRChain is a smart-contract based system for exchanging health data based on the standard FHIR (Fast Healthcare Interoperability Resources) [33], where clinical data is stored off chain, and the blockchain itself stores encrypted meta-data which serve as pointers to the primary data source (like an Electronic Health Record - EHR). Azaria et al. introduced MedRec [3], which uses a permissioned blockchain network to facilitate data sharing and authentication. MedRec has a novel proof-of-work incentive method built around access to anonymized medical data. Dubovitskaya et al. also propose a permissioned blockchain (focused on oncologic care) which leverages off-chain cloud storage for clinical data, using the blockchain to manage consent and authorization [23]. MedShare [61] engineered their own blockchain implementation to implement a system for secure medical data sharing among big data entities. This goal is achieved by utilizing smart contracts and access control mechanisms. Finally, MIStore [66] addressed the hospital-patient-insurance problem with a blockchain-based medical insurance storage system. They introduced a system with one hospital, one patient, one insurance company and n servers. The privacy-preserving property is achieved with (t,n) threshold protocol. They also have verifiable data and commitment in every step. However, they also have some weaknesses: If the number of servers gets large, the performance will be affected. Furthermore, the system may leak patient information if the hospital reuse the cryptographic key for multiple patients.

Chapter 3

Cryptographic Background

In this chapter, we first discuss some cryptographic background, including bilinear pairing of prime order, digital signature, commitment, and secret sharing techniques, which will be used for our proposed scheme.

3.1 Bilinear Groups of Prime Order

Bilinear pairing is a popular tool that has been widely used in cryptographic systems [14, 15, 45, 13]. Let $\mathbb{G}$ and $\mathbb{G}_T$ be cyclic multiplicative groups of prime order q. A multiplicative group of prime order q is the set of non-negative integers coprime to q that form a group under multiplication modulo q. A cyclic group can be generated by a single element from the group. A bilinear pairing is a map:

$$e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$$

satisfying the following properties:

1. **Bilinearity:** For any $a, b \in Z_q^*$ and $P, Q \in \mathbb{G}$, we have $e(P^a, Q^b) = e(P, Q)^{ab}$.

2. **Non-degeneracy:** There exists $P, Q \in \mathbb{G}$ such that $e(P, Q) \neq 1$.

3. **Computability:** There exists an efficient algorithm to compute $e(P, Q)$ for all $P, Q \in \mathbb{G}$.

Definition 3.1.1 (Bilinear Generator)**.** *A bilinear parameter generator Gen is a probabilistic algorithm that takes a security parameter κ as input and outputs a 5-tuple $(q, P, \mathbb{G}, \mathbb{G}_{\mathbb{T}}, e)$, where q is a κ-bit prime number, ($\mathbb{G}$, $\times$) and ($\mathbb{G}_T$, $\times$) are two groups with the same order q, $P \in G$ is a generator,and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is an admissible bilinear map.*

Next, we define the quantitative notion of the complexity assumptions, including Computational Diffie-Hellman (CHD) Problem, Decisional Diffie-Hellman (DDH) Problem, Bilinear Diffie-Hellman (BDH) Problem, Decisional Diffie-Hellman (DBDH) Problem, Strong Diffie-Hellman (SDH) Problem, and Strong Diffie-Hellman-2 (SDH2) Problem in groups $\mathbb{G}$ and $\mathbb{G}_T$.

Definition 3.1.2 (Computational Diffie-Hellman (CDH) Problem)**.** *The Computational Diffie-Hellman (CDH) problem in $\mathbb{G}$ is defined as follows: Given $P, P^a, P^b \in \mathbb{G}$ for unknown $a, b \in Z_q^*$, compute $P^{ab} \in \mathbb{G}$.*

Definition 3.1.3 (CDH Assumption)**.** *Let A be an adversary that takes an input of $(P, P^a, P^b) \in \mathbb{G}$ for unknown $a, b \in Z_q^*$, and returns P^{ab}. We consider the following random experiment.*

$$\begin{array}{l}\texttt{Experiment}\ \mathbf{Exp}_{\mathcal{A}}^{CDH} \\ \quad a, b \xleftarrow{R} \mathbb{Z}, \alpha \leftarrow \mathcal{A}(P, P^a, P^b) \\ \quad \texttt{if}\ \alpha = P^{ab},\ \texttt{then}\ \beta \leftarrow 1,\ \texttt{else}\ \beta \leftarrow 0 \\ \quad \texttt{return}\ \beta\end{array}$$

We define the corresponding success probability of $\mathcal{A}$ in solving the CDH problem via

$$\mathbf{Succ}_{\mathcal{A}}^{\mathrm{CDH}} = \Pr\left[\mathbf{Exp}_{\mathcal{A}}^{\mathrm{CDH}} = 1\right]$$

Let $\tau \in \mathbb{N}$ and $\epsilon \in [0, 1]$. We say that the CDH is (τ, ϵ)-secure if no polynomial algorithm $\mathbb{A}$ running in time τ has success $\mathbf{Succ}_{\mathcal{A}}^{\mathrm{CDH}} \geq \epsilon$

Definition 3.1.4 (Decisional Diffie-Hellman (DBDH) Problem)**.** *The Decisional Bilinear Diffie-Hellman (DBDH) problem in $\mathbb{G}$ is as follows: Given an element P of $\mathbb{G}$, a tuple (P^a, P^b, P^c, T) for unknown $a, b, c \in \mathbb{Z}_q^*$ and $T \in \mathbb{G}_T$, decide whether $T = e(P, P)^{abc}$ or a random element R drawn from $\mathbb{G}_T$.*

Definition 3.1.5 (DBDH Assumption)**.** *Let $\mathcal{A}$ be an adversary that takes an input of (P^a, P^b, P^c, T) for unknown $a, b, c \in \mathbb{Z}_q^*$ and $T \in \mathbb{G}_T$, and returns a bit $\beta' \in \{0,\ 1\}$. We consider the following random experiments.*

Experiment $\mathbf{Exp}_{\mathcal{A}}^{DBDH}$

$a, b, c \xleftarrow{R} \mathbb{Z}_q;\ , R \xleftarrow{R} \mathbb{G}_T$

$\widetilde{\beta} \leftarrow \{0,\ 1\}$

if $\widetilde{\beta} = 0$, then $T = e(P,\ P)^{abc}$; else if $\widetilde{\beta} = 1$ then $T = R$

$\widetilde{\beta'} \leftarrow \mathcal{A}(P^a, P^b, P^c, T)$

return 1 if $\widetilde{\beta'} = \widetilde{\beta}$, 0 otherwise

We then define the advantage of $\mathcal{A}$ via

$$\mathbf{Adv}_{\mathcal{A}}^{DBDH} = \left|\Pr\left[\mathbf{Exp}_{\mathcal{A}}^{DBDH} = 1|\widetilde{\beta} = 0\right] - \Pr\left[\mathbf{Exp}_{\mathcal{A}}^{DBDH} = 1|\widetilde{\beta} = 1\right]\right| \geq \epsilon$$

Let $\tau \in \mathbb{N}$ and $\epsilon \in [0,\ 1]$. We say that the DBDH is (τ, ϵ)-secure if no adversary $\mathcal{A}$ running in time τ has an advantage $\mathbf{Succ}_{\mathcal{A}}^{CDH} \geq \epsilon$.

Definition 3.1.6 (Strong Diffie-Hellman (SDH) Problem)**.** *The Strong Diffie-Hellman (CDH) problem in $\mathbb{G}$ is defined as follows: Given an element P of $\mathbb{G}$, and an l-tuple $(P^x, P^{x^2}, \cdots, P^{x^l})$ for some unknown $x \in \mathbb{Z}_q$, compute a new tuple $(c, P^{\frac{1}{x+c}})$, where $c \in \mathbb{Z}_q$.*

Definition 3.1.7 (Strong Diffie-Hellman (SDH) Assumption)**.** *Let $\mathcal{A}$ be an adversary that takes an input of $(P^x, P^{x^2}, \cdots, P^{x^l})$ for some unknown $x \in \mathbb{Z}_q$, and returns a new tuple $(c, P^{\frac{1}{x+c}})$. We consider the following random experiment.*

$\texttt{Experiment}\ \mathbf{Exp}_{\mathcal{A}}^{SDH}$

$x \xleftarrow{R} \mathbb{Z}_q, (c, \alpha) \leftarrow \mathcal{A}(P, P^x, P^{x^2}, \cdots, P^{x^l})$

$\texttt{if}\ \alpha = P^{\frac{1}{x+c}}\ \texttt{then}\ \beta \leftarrow 1,\ \texttt{else}\ \beta \leftarrow 0$

$\texttt{return}\ \beta$

We define the corresponding success probability of $\mathcal{A}$ in solving the SDH problem via

$$\mathbf{Succ}_{\mathcal{A}}^{\mathrm{SDH}} = \Pr\left[\mathbf{Exp}_{\mathcal{A}}^{\mathrm{SDH}} = 1\right]$$

Let $\tau \in \mathbb{N}$ and $\epsilon \in$ [0, 1]. We say that the SDH is (τ, ϵ)-secure if no polynomial algorithm $\mathbb{A}$ running in time τ has success $\mathbf{Succ}_{\mathcal{A}}^{\mathrm{SDH}} \geq \epsilon$

Definition 3.1.8 (Strong Diffie-Hellman-2 (SDH2) Problem)**.** *The strong Diffie-Hellman-2 (SDH2) problem in $\mathbb{G}$ is defined as follows: Let P be an element of $\mathbb{G}$. Given $(P^x, P^y, P^{\frac{1}{x+y}})$ for unknown $x, y \in \mathbb{Z}_q$, l_1 distinct tuples $(c_i, P^{\frac{1}{x+c_i}})$, where $c_i \in \mathbb{Z}_q, i \in \{1, 2, \cdots, l_1\}$, and another tuple $(P^{y^2}, \cdots, P^{y^{l_2}})$, compute a new tuple $(m, P^{\frac{1}{y+m}})$, where $m \in \mathbb{Z}_q$.*

Definition 3.1.9 (SDH2 Assumption)**.** *Let $\mathcal{A}$ be an adversary that takes an input of $P, P^x, P^y, P^{\frac{1}{x+y}}, c_1, P^{\frac{1}{x+c_1}}, c_2, P^{\frac{1}{x+c_2}}, \cdots, c_{l_1}, P^{\frac{1}{x+c_{l_1}}}$ for some unknown $x, y, c_1, \cdots, c_{l_1} \in \mathbb{Z}_q$, and another tuple $(P^{y^2}, \cdots, P^{y^{l_2}})$, returns a new tuple (c, α). We consider the following random experiment.*

Experiment $\mathbf{Exp}_{\mathcal{A}}^{SDH2}$

$$x, y, c_1, \cdots, c_{l_1} \xleftarrow{R} \mathbb{Z}_q$$

$$(m, \alpha) \leftarrow \mathcal{A}\left(\begin{array}{c} P, P^{y^{l_2}}, P^{y^{l_2}}, P^{\frac{1}{x+y}} \\ c_1, P^{\frac{1}{x+c_1}}, c_2, P^{\frac{1}{x+c_2}}, \cdots, c_{l_1}, P^{\frac{1}{x+c_{l_1}}} \\ P^{y^2}, \cdots, P^{y^{l_2}} \end{array}\right)$$

`if` $\alpha = P^{\frac{1}{y+m}}$ `then` $\beta \leftarrow 1$, `else` $\beta \leftarrow 0$

`return` β

We define the corresponding success probability of $\mathcal{A}$ in solving the SDH2 problem via

$$\mathbf{Succ}_{\mathcal{A}}^{\mathrm{SDH2}} = \Pr\left[\mathbf{Exp}_{\mathcal{A}}^{\mathrm{SDH2}} = 1\right]$$

Let $\tau \in \mathbb{N}$ and $\epsilon \in$ [0, 1]. We say that the SDH2 is (τ, ϵ)-secure if no polynomial algorithm $\mathbb{A}$ running in time τ has success $\mathbf{Succ}_{\mathcal{A}}^{\mathrm{SDH2}} \geq \epsilon$

3.2 Digital Signatures

Digital signature is a way to guarantee the integrity and authenticity of an electronic message. When a message is digitally signed, the receiver of the message can use the signature to verify that the sender is real and the message has not been altered.

Digital signatures are based on public-key cryptography and hash functions, where the sender generates a hash of the message, and then encrypt it with their private key. The receiver, upon receiving the signed message, can use the sender's public key together with the hash function to verify if the signature matches the message, as shown in Figure 3.1.

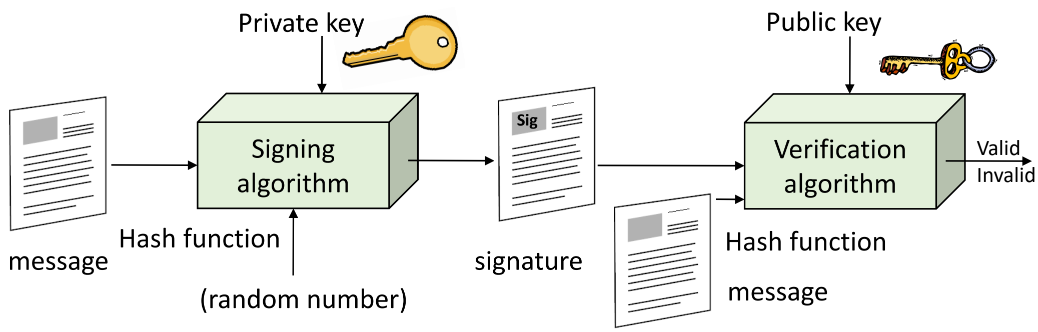

Figure 3.1: The digital signature technique

In our scheme, we will use the short signature scheme from bilinear pairing proposed by Zhang et al. [64]. The system parameters of the short signature scheme are as follows: Let $\mathbb{G}$ and $\mathbb{G}_T$ be cyclic multiplicative groups of prime order q. The bilinear pairing is $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. The cryptographic hash function H is defined $H : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$. The scheme consists of the following four algorithms: a parameter generation algorithm `ParamGen`, a key generation algorithm `KeyGen`, a signature generation algorithm `Sign` and a signature verification algorithm `Ver`, which are described in detail as follows:

1. `ParamGen`. The system parameters are $\{\mathbb{G}, \mathbb{G}_T, e, q, P, H\}$.

2. `KeyGen`. Randomly selects $x \in {}_R\mathbb{Z}_q^*$, and computes $P_{pub} = P^x$. The public key is P_{pub}. The private key is x.

3. $\texttt{Sign}$. Given a secret key x, and a message m, computes the signature $S = P^{\frac{1}{H(m)+x}}$.

4. $\texttt{Ver}$. Given a public key P_{pub}, a message m, and a signature S, verify if

$$e(P^{H(m)} \cdot P_{pub}, S) = e(P, P).$$

The verification works because of the following equations:

$$\begin{aligned} e(P^{H(m)} \cdot P_{pub}, S) &= e(P^{(H(m)+x)}, P^{(H(m)+x)^{-1}}) \\ &= e(P, P)^{(H(m)+x)\cdot(H(m)+x)^{-1}} \\ &= e(P, P) \end{aligned}$$

According to [64], the short signature is secure against an adaptive chosen message attack under the random oracle model [10].

3.3 Commitment Technique

Commitment is an important cryptographic technique, which has been widely used in various privacy-preserving applications [54]. A commitment protocol between two entities - Alice and Bob - consists of two phases: `commit` and `open`. In the `commit` phase, Alice creates and sends Bob a commitment $c = \texttt{Com}(m, r)$ for a message m, where r is a random number. In the `open` phase, Alice reveals (m, r) and Bob can calculate $c' = \texttt{Com}(m, r)$. If $c = c'$ then Bob can be sure that the value of m and r is correct. $\text{Com}(m, r)$ need to satisfy two properties: *hiding* and *binding*. *Hiding* means that it should be impossible for Bob to figure out the value of m from c, while *Binding* means that Alice cannot change the value of m after c is generated. From existing literature, we can find that these two properties can be achieved computationally, statistically or perfectly [54].

We will present some background definitions of commitments that are used in blockchains [17][51]. In what follows, a Probabilistic Polinomial Time (PPT) adversary $\mathcal{A}$ is a probabilistic interactive Turing Machine that runs in polynomial time in the security parameter λ.

Definition 3.3.1 (Discrete Log Relation). *For all PPT adversaries $\mathcal{A}$ and for all $n \geq 2$ there exists a negligible function $\mu(\lambda)$ such that*

$$P\left[\begin{matrix}\mathbb{G} = \mathrm{Setup}(1^\lambda), g_1, \cdots, g_n \xleftarrow{\$} \mathbb{G}; \\ a_1, \cdots, a_n \in \mathbb{Z}_p \leftarrow \mathcal{A}(G, g_1, \cdots, g_n)\end{matrix} : \exists a_i \neq 0 \wedge \prod_{i=1}^{n} g_i^{a_i} = 1\right] \leq \mu(\lambda)$$

Definition 3.3.2 (Commitment). *A non-interactive commitment scheme consists of a pair of probabilistic polynomial time algorithms* (Setup, Com)*. The setup algorithm* $\mathrm{pp} \leftarrow \mathrm{Setup}(1^\lambda)$ *generates public parameters* pp *for the scheme, for security parameter λ. The commitment algorithm* $\mathrm{Com}_{\mathrm{pp}}$ *defines a function* $\mathrm{M}_{\mathrm{pp}} \times \mathrm{R}_{\mathrm{pp}} \rightarrow \mathrm{C}_{\mathrm{pp}}$ *for message space* M_{pp}*, randomness space* R_{pp} *and commitment space* C_{pp} *determined by* pp*. For a message $x \in \mathrm{M}_{\mathrm{pp}}$, the algorithm draws $r \xleftarrow{\$} \mathrm{R}_{\mathrm{pp}}$ uniformly at random, and computes commitment* $\mathbf{com} = \mathrm{Com}_{\mathrm{pp}}(x; r)$*. For ease of notation we write* $\mathrm{Com} = \mathrm{Com}_{\mathrm{pp}}$*.*

Definition 3.3.3 (Homomorphic Commitments)**.** *A homomorphic commitment scheme is a non-interactive commitment scheme such that* M_{pp}*,* R_{pp} *and* C_{pp} *are all abelian groups, and for all* $x_1, x_2 \in \mathrm{M}_{\mathrm{pp}}, r_1, r_2 \in \mathrm{R}_{\mathrm{pp}}$*, we have*

$$\mathrm{Com}(x_1; r_1) + \mathrm{Com}(x_2, r_2) = \mathrm{Com}(x_1 + x_2; r_1 + r_2)$$

Definition 3.3.4 (Hiding Commitment)**.** *A commitment scheme is said to be hiding if for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* $\mu(\lambda)$ *such that*

$$\left| \mathrm{P}\left[b = b' \left| \begin{array}{l} \mathrm{pp} \leftarrow \mathrm{Setup}(1^\lambda); \\ (x_0, x_1) \in \mathrm{M}_{\mathrm{pp}}^2 \leftarrow \mathcal{A}(\mathrm{pp}), b \xleftarrow{\$} \{0,1\}, r \xleftarrow{\$} \mathrm{R}_{\mathrm{pp}}, \\ \mathbf{com} = \mathrm{Com}(x_b; r), b' \leftarrow \mathcal{A}(\mathrm{pp}, \mathbf{com}) \end{array} \right. \right] - \frac{1}{2} \right| \leq \mu(\lambda)$$

Definition 3.3.5 (Binding Commitment)**.** *A commitment scheme is said to be binding if for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* μ *such that*

$$\mathrm{P}\left[\mathrm{Com}(x_0; r_0) = \mathrm{Com}(x_1; r_1) \wedge x_0 \neq x_1 \left| \begin{array}{l} \mathrm{pp} \leftarrow \mathrm{Setup}(1^\lambda), \\ x_0, x_1, r_0, r_1 \leftarrow \mathcal{A}(\mathrm{pp}) \end{array} \right. \right] \leq \mu(\lambda)$$

where the probability is over Setup *and* $\mathcal{A}$*. If* $\mu(\lambda) = 0$ *then we say the scheme is* perfectly binding.

Below, the order p of the groups used is implicitly dependent on the security parameter λ to ensure that discrete log in these groups is intractable for PPT adversaries.

Definition 3.3.6 (Pedersen Commitment)**.** Setup*: The receiver chooses large primes p and q such that $q|p-1$, a generator g of the order-q subgroup of $\mathbb{Z}_p^*$ and a random secret $a \in \mathbb{Z}_q$. Calculate $h = g^a \mod p$. The values (p, q, g, h) are public and a is secret.*
Commit*: To commit to some $x \in \mathbb{Z}_q$, the sender chooses a random $r \in \mathbb{Z}_q$ and sends $c = g^x h^r \mod p$ to the receiver. Note that because $h = g^a \mod p$, we have $c = g^x (g^a)^r = g^{x+ar} \mod p$.*
Reveal*: To open the commitment, the sender reveals x and r, the receiver verifies that $c = g^x h^r \mod p$ The Pedersen Commitment is perfectly hiding and computationally binding.*

- *Perfectly hiding: Given commitment c, every value x is equally likely to be the value committed in c. Suppose we have x, r and any x', there exists r' such that $g^x h^r = g^{x'} h^{r'}$. Knowing a, we can simply calculate $r' = (x - x')a^{-1} + r \mod q$.*

- *Computationally binding: If the sender can find different x and x' both of which open commitment $c = g^x h^r$, it means he can find r' such that $g^x h^r = g^{x'} h^{r'} \mod p$, or $g^{x+ar} = g^{x'+ar'} \mod p$. If the sender is able to compute $a = \log_g h \mod p$ then he can solve $a = \frac{x-x'}{r-r'}$, but then it means he can solve the discrete logarithm problem.*

3.4 Secret Sharing Technique

The problem of secret sharing arises when a group of n participants want to share a secret s in such a way that the secret can be recovered once you have a certain amount of fragments. The most popular secret sharing scheme is Shamir's Secret Sharing Scheme (SSSS) [55], which is explained below.

3.4.1 (t,n)-Secret Sharing Technique

The goal of (t, n)-Secret Sharing is to divide a secret s in to n pieces $s_1, \cdots, s_n$ in such a way that: i) Knowledge of any t or more s_i pieces make s easily computable; and ii) knowledge of any $t-1$ or fewer s_i pieces leaves s completely undetermined. In the following, we first consider $t = 2, 3$ and review $(2, n)$-secret sharing technique, $(3, n)$-secret sharing. After that, we recall Shamir's (t, n)-secret sharing technique.

3.4.1.1 (2,n)-Secret Sharing Technique

In order to share a secret s among n parties, where any two shares can recover the secret, we can use a $(2, n)$-secret sharing technique. Using basic geometry, we know that two points are sufficient to define a line. By drawing a line that intersects with the y axis at s, as long as we have two points along that line, we can always recover the line and the secret.

3.4.1.2 (3,n)-Secret Sharing Technique

In order to share a secret s among n parties where any three shares can recover the secret, we can use a $(3, n)$-secret sharing technique. Similar to $(2, n)$-secret sharing technique, some basic geometry can be used. We know that three points are sufficient to define a parabola. By drawing a parabola that intersects with the y axis at s, any point in the parabola can be used as a share. As long as we have three points, we can recover the parabola and the secret.

3.4.1.3 Shamir's (t,n)-Secret Sharing Technique

Clearly, the above $(3, n)$-secret sharing technique is a generalization of $(2, n)$-secret sharing technique. If we keep going with higher degree polynomials, we will gain the Shamir's (t, n)-Secret Sharing Technique. Concretely, Shamir's (t, n)-Secret Sharing Technique is described as follows:

First, choose a large prime p. In order to share a secret x, i.e., $x \Rightarrow (x_1, x_2, \cdots, x_n)$, the following steps will be run:

- Choose coefficients $f_1, \cdots, f_{t-1} \in \mathbb{Z}_p^*$, which are to be the coefficients of degree $t-1$ polynomial f.
- Let $f(z) = f_0 + f_1 \cdot z + \cdots f_{t-1} \cdot z^{t-1}$, where $f_0 = x$.
- Give $f(i)$ to party i, $i = 1, \cdots, n$.

When t parties $\mathcal{P}$ work together, we can use Lagrange Interpolation to recover the secret f_0. Because, given (i, x_i) for $i \in \mathcal{P}$,

$$f(z) = \sum_{i \in \mathcal{P}} x_i \left(\prod_{j \in \mathcal{P}, j \neq i} \frac{z - j}{i - j} \right)$$

is a linear system in t unknowns, the coefficients f_k, with t equations. Then, the secret x can be recovered as

$$x = f_0 = f(0) = \sum_{i \in \mathcal{P}} x_i \left(\prod_{j \in \mathcal{P}, j \neq i} \frac{-j}{i - j} \right)$$

3.4.2 Our (2,3)-Secret Sharing Scheme

	$s1$	$s2$	$s3$
n_1	✓	✓	
n_2		✓	✓
n_3	✓		✓
$n_1 \cup n_2$	✓	✓	✓
$n_2 \cup n_3$	✓	✓	✓
$n_3 \cup n_1$	✓	✓	✓

Table 3.1: Our $(2, 3)$-Secret Sharing Scheme

In order to fit our problem definition, we consider a $(2, 3)$-Secret Sharing Scheme, which can be described as follows. Let s be the secret to be shared among 3 parties (n_1, n_2, n_3), and any two shares can recover the secret. In particular, the secret s can be divided into (s_1, s_2, s_3) where $s = s_1 + s_2 + s_3$. Then (s_1, s_2), (s_2, s_3) and (s_3, s_1) are sent to n_1, n_2, n_3, respectively. Later, when any 2 parties collaborate, the secret s can be recovered, as shown in Table 3.1.

3.5 Summary

In this chapter, we have discussed some cryptographic background, including bilinear groups of prime order, digital signature, commitment, and secret sharing techniques, which will be used in our proposed scheme in next chapters.

Chapter 4

Models and Design Goals

In this chapter, we formalize our system model, security model, and identify our design goals for a a blockchain-based medical insurance storage system that is secured, verifiable and privacy-preserving.

4.1 System Model

In our system model, we consider a blockchain-based storage and query service over insurance data, which is comprised of five types of entities: a public blockchain, a data owner HS_0, three helpers HS_1, HS_2, HS_3, a patient and an insurance company. The role of data owner and helpers are played by hospitals.

- Data owner HS_0: The data owner HS_0 is responsible for initializing

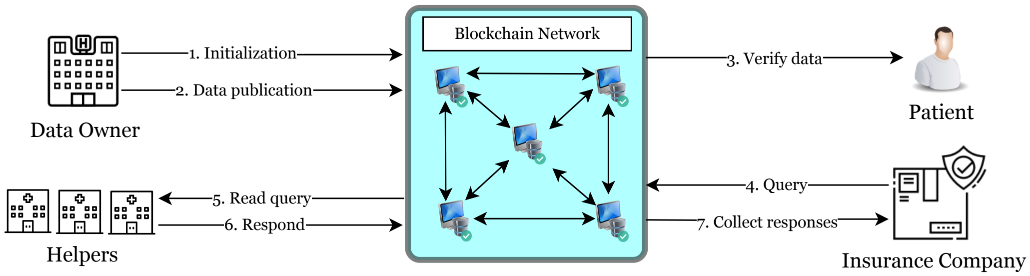

Figure 4.1: Full storage and query process

the system and publish patients' spending records on the blockchain. Each record d_z on the blockchain is the value of an invoice for a single hospital visit. The data owner generates and assigns keys for helpers HS_1, HS_2, HS_3. Later, the data will be encrypted and published on the blockchain in such a way that the data requires the collaboration of any two helpers to recover.

- Helper Hospitals HS_1, HS_2, HS_3: In our model, the role of helper hospitals HS_1, HS_2, HS_3 is to help the insurance company to recover the spending records of a patient. For any invoice ID_z, if the invoice does not originate from a hospital, then that hospital is a helper. The helpers monitor the blockchain for data requests from insurance company I and process the requests separately, each helper will publish their part of the answer to the blockchain. The insurance company only needs to get the answer from 2 helpers to recover the data. This model can ensure high availability from the helpers.

- Blockchain: In our model, the blockchain is a public, permissionless blockchain. The public key of the hospitals must be publicly available for everyone to verify the authenticity of the data. We use Ethereum for the implementation of this system, as Ethereum possesses a Turing-complete language suitable for creating complex applications on the blockchain. Each entity in our system will communicate through the blockchain and there is no direct communication between entities.

- Patient: In our model, the patient P_i is anybody who has spent money for hospital service. The patient does not play any role in data publication and retrieval, but they can verify the numbers published by the data owner, and complain if the numbers are wrong.

- Insurance Company: In our model, the insurance company I is interested in knowing the total spending of a patient for insurance purpose. I cannot know the details of each invoice, but it can send to the blockchain a query for the total amount of money patient P_i has spent in hospital HS_0. After the helper hospitals have processed the query and published the result on the blockchain, I can query the blockchain for the requested data.

In the following, we will give the data publication and query process in detail. The process in run among the data owner, helpers, patient and insurance company, with each action recorded on the blockchain, as shown in Figure 4.1. The hospitals HS_0, HS_1, HS_2, HS_3 and insurance company I register themselves on the blockchain and make their public keys known to other parties. HS_0 then initializes the system by generating the keys x_1, x_2, x_3 and assign them to HS_1, HS_2, HS_3. Later on, when HS_0 wants to publish some value d_z, he/she needs to split d_z into $d_{z_1}, d_{z_2}, d_{z_3}$ and encrypt the values with x_1, x_2, x_3. When insurance companies I sends a query for patient P's spending records, the helpers will calculate $d_{z_1}, d_{z_2}, d_{z_3}$. I can then combine the results to get the total amount of money the patient has spent.

4.2 Security Model

4.2.1 Security Assumptions

In our security model, the data owner HS_0 is considered to be $partiallyhonest$, i.e. it will follow the protocol to encrypt the patient's invoice data and publish them to the blockchain, but the data sent by them should be verified. The helpers HS_1, HS_2, HS_3 are also considered $honest - but - curious$, in other words, they will take the query request from the insurance company and honestly process and return the data as requested, but they may also be interested in the details of individual invoices. The protocol is considered secure if at least 2/3 of the helpers are *honest*. The insurance company is $honest - but - curious$, they follow the protocol to get a patient's total spending, but also interested in the details of individual invoices. The blockchain is considered $tamper - resistant$ and all data on the blockchain is *untampered.* In our system, most data published on the blockchain is verifiable. Because every transaction on the blockchain must be signed by the author, impersonation attacks are considered to be impossible.

4.2.2 Security Requirements

Keeping a patient's data secure and private is always vital in healthcare systems. Without the guarantee of a patient's privacy, a heath data storage application cannot be widely accepted by the public. For that reason, it's essential to protect the patient data. Specifically, the following security requirements should be ensured: i) the patient data should be protected from unauthorized entities; ii) the data accuracy should be guaranteed; iii) the data should be protected from possible system failures.

4.3 Design Goals

Our design goal is to achieve privacy-preserving query over encrypted insurance data on the blockchain. The system should also be verifiable and has high-availability. Specifically, the following objectives should be satisfied.

- Privacy Preservation: The most important requirement of this scheme is privacy preservation. In other words, the initialization data, the stored spending records and the query result on the blockchain are kept secret from unauthorized entities.

- Verifiable: Every item published on the blockchain is publicly verifiable to prevent from cheating. Furthermore, a mechanism is also implemented to punish the data owner or helpers if they try to provide incorrect information. Because every transaction is digitally signed, the system is also non-repudiable.

- High-availability: Because every communication happens on the blockchain, the system is required to have high availability. In our case, we employ the $(2, 3)$-secret sharing scheme which is robust against failures as long as 2/3 helpers function properly. Generally, the system can be scaled to support any (t, n) when more hospitals join the network.

Chapter 5

Proposed Blockchain-based Privacy-preserving Medical Insurance Storage System

In this chapter, we present our blockchain-based privacy-preserving medical insurance storage system, which mainly consists of 7 steps, namely *Initialization, Data Publication, Data Verification, Query, Respond*, and *Response Collection*.

For a better description, we denote the owner hospital as HS_0, the helper hospitals are HS_1, HS_2 and HS_3. P is the patient and I is the insurance company. Each time patient P pays for hospital services and receives an invoice Z, meanwhile he will also receive a secret key S_z from the owner hospital HS_0, which can be used to check the value of the invoice in the future. Note that, the secret key S_z is only known by the owner hospital HS_0 and patient P. In addition, there could be multiple patients and insurance companies in our system, but for the sake of simplicity, we just consider one patient P in our scheme description. Before describing the details of our scheme, some cryptographic parameters are first listed in Table 5.1.

Symbol	Description
$\mathbb{G}$	A multiplicative cyclic group
g	The generator of $\mathbb{G}$
q	The order of $\mathbb{G}$
e	The bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$
H	A hash function that performs the mapping $H : \{0,1\}^* \to \mathbb{G}$
ID_{HS_0}	ID of the owner hospital
ID_{H_i}	ID of the helper hospital i
ID_P	ID of the patient
ID_I	ID of the insurance company
ID_z	ID of a single invoice Z
d_z	The value of a single invoice Z
d_{z_i}	Part i of invoice Z, which is splitted into 3 parts
pk_i, sk_i	Public / private key pair of hospital i, owner hospital has $i = 0$.
pk_P, sk_P	Public / private key pair of the patient
pk_I, sk_I	Public / private key pair of the insurance company

Table 5.1: Symbols used in the scheme

5.1 Security Deposit

Before the initialization stage, the data owner HS_0 and helper hospitals HS_1, HS_2, HS_3 publish smart contracts to hold some coins of the blockchain as security deposit. If later some party publishes incorrect data that can be verified, the evidence can be sent to the publisher's smart contract to claim some rewards. The security of smart contract depends on the security mechanism of blockchain. Only if the employed blockchain platform is secure, the security deposit can be guaranteed. After the security deposit stage, the owner hospital HS_0 initializes the system.

5.2 Initialization

The owner hospital HS_0 needs to generate keys for the helper hospitals HS_1, HS_2 and HS_3. In order to do that, HS_0 generates the parameters $(a_1,\, a_2,\, a_3,\, b_1,\, b_2,\, b_3,\, x_1,\, x_2,\, x_3) \in \mathbb{Z}_q^*$ such that

$$\begin{cases} a_1b_1 = a_2 + b_2 + x_1 \bmod q \\ a_2b_2 = a_3 + b_3 + x_2 \bmod q \\ a_3b_3 = a_1 + b_1 + x_3 \bmod q \end{cases} \tag{5.1}$$

After that, the owner hospital HS_0 calculates the commitments $(g^{a_1},\, g^{a_2},\, g^{a_3},\, g^{b_1},\, g^{b_2},\, g^{b_3},\, g^{x_1},\, g^{x_2},\, g^{x_3})$, and $(x_1,\, x_2,\, x_3)$ are respectively encrypted with helper hospital HS_1, HS_2, HS_3's public keys to generate the ciphertexts $E_{pk_1}(x_1)$, $E_{pk_2}(x_2)$, $E_{pk_3}(x_3)$. Finally, the following committed data and keys $\mathbb{D}_{init}$ will be published on the blockchain.

$$\mathbb{D}_{init} = \begin{bmatrix} g^{a_1}, g^{a_2}, g^{a_3} \\ g^{b_1}, g^{b_2}, g^{b_3} \\ g^{x_1}, g^{x_2}, g^{x_3} \\ E_{pk_1}(x_1), E_{pk_2}(x_2), E_{pk_3}(x_3) \end{bmatrix}$$

Key Verification. To verify the correctness of keys (x_1, x_2, x_3) in $\mathbb{D}_{init}$, everyone can check :

$$\begin{cases} e(g^{a_1}, g^{b_1}) \stackrel{?}{=} e(g^{a_2} \cdot g^{b_2} \cdot g^{x_1}, g) \\ e(g^{a_2}, g^{b_2}) \stackrel{?}{=} e(g^{a_3} \cdot g^{b_3} \cdot g^{x_2}, g) \\ e(g^{a_3}, g^{b_3}) \stackrel{?}{=} e(g^{a_1} \cdot g^{b_1} \cdot g^{x_3}, g) \end{cases} \tag{5.2}$$

If all of them are true, the correctness of (x_1, x_2, x_3) can be confirmed, otherwise, they will not be accepted, because

$$\begin{cases} e(g^{a_1}, g^{b_1}) = e(g,g)^{a_1 b_1} = e(g,g)^{a_2+b_2+x_1} = e(g^{a_2} \cdot g^{b_2} \cdot g^{x_1}, g) \\ e(g^{a_2}, g^{b_2}) = e(g,g)^{a_2 b_2} = e(g,g)^{a_3+b_3+x_2} = e(g^{a_3} \cdot g^{b_3} \cdot g^{x_2}, g) \\ e(g^{a_3}, g^{b_3}) = e(g,g)^{a_3 b_3} = e(g,g)^{a_1+b_1+x_3} = e(g^{a_1} \cdot g^{b_1} \cdot g^{x_3}, g) \end{cases} \tag{5.3}$$

For the security of keys (x_1, x_2, x_3), we can see that the published data $\mathbb{D}_{init}$ actually is a commitment scheme, which is not only *computationally hiding* but also *computationally binding* under the discrete logarithm assumption.

- *Computationally Hiding.* Because the discrete logarithm assumption, given g^{x_i}, it is hard for an attacker to obtain x_i. As a result, it is *computationally hiding*.

- *Computationally Binding.* At the same time, given g^{a_1}, g^{a_2}, g^{a_3}, g^{b_1}, g^{b_2}, g^{b_3}, g^{x_1}, g^{x_2}, g^{x_3}, it is hard for an attacker to find another pair of (x'_1, x'_2, x'_3) such that $x'_1 \neq x_1, x'_2 \neq x_2, x'_3 \neq x_3$ and Eq. (5.2) is still true. Actually, based on Eq. (5.1), we can see (x_1, x_2, x_3) are uniquely determined by $(a_1, b_1, a_2, b_2, a_3, b_3)$, without knowing the latter (from the discrete logarithm assumption), (x_1, x_2, x_3) cannot be changed. Therefore, the published data $\mathbb{D}_{init}$ is also *computationally binding*.

Note that, every one can verify the correctness of the keys (x_1, x_2, x_3) from the commitment $\mathbb{D}_{init}$, but cannot gain the keys. While, each helper hospital HS_i, $i \in \{1, 2, 3\}$, can use its private key sk_i to recover x_i from $E_{pk_i}(x_i)$, and verify x_i with the committed g^{x_i}.

5.3 Data Publication

Consider patient P who has an invoice Z in the owner hospital HS_0, the value of the invoice Z is d_z. The goal of HS_0 is to securely publish the value d_z to the blockchain in such a way that nobody can know d_z without the assistance from the helper hospitals HS_1, HS_2, HS_3. In order to achieve secrecy, the owner hospital HS_0 employs a $(2, 3)$ secret sharing scheme that splits the value d_z in to three secret values $(d_{z_1}, d_{z_2}, d_{z_3})$ such that $d_{z_1}, d_{z_2}, d_{z_3} \in \mathbb{Z}_q^*$ and

$$d_z = d_{z_1} + d_{z_2} + d_{z_3} \bmod q \tag{5.4}$$

Then, these values will be securely distributed to HS_1, HS_2, HS_3, as illustrated in Table 5.2.

	d_{z_1}	d_{z_2}	d_{z_3}
HS_1	✓	✓	
HS_2		✓	✓
HS_3	✓		✓

Table 5.2: Distribution scheme for invoice value parts. HS_1 can read d_{z_1}, d_{z_2}; HS_2 can read d_{z_2}, d_{z_3} and HS_3 can read d_{z_3}, d_{z_1}.

Because all communications are published on the public blockchain, the owner hospital HS_0 needs a way to encrypt $(d_{z_1}, d_{z_2}, d_{z_3})$ so that only the intended recipients can read the corresponding data. To achieve that goal, those values $(d_{z_1}, d_{z_2}, d_{z_3})$ need to be encrypted with parameters only known to the helper hospitals HS_1, HS_2 and HS_3, i.e., the respective keys x_1, x_2, x_3 generated from Eq. (5.1). For example, in order to encrypt d_{z_1} into a ciphertext $d_{z_1,1}$ for HS_1 to read, we can use the following equation:

$$d_{z_1,1} = H(g^{a_1 \cdot H(x_1 || ID_z)}) \cdot d_{z_1} \bmod q \tag{5.5}$$

Note that, HS_1 knows the value of x_1, and g^{a_1} has already been published on the blockchain with $\mathbb{D}_{init}$, so HS_1 can calculate $H(g^{a_1 \cdot H(x_1 || ID_z)})$ without any troubles. Then with $d_{z_1,1}$ and $H(g^{a_1 \cdot H(x_1||ID_z)})$, it's trivial to retrieve d_{z_1} by the following equation

$$d_{z_1} = d_{z_1,1} \cdot \frac{1}{H(g^{a_1 \cdot H(x_1||ID_z)})} \mod q \tag{5.6}$$

Following the same logic demonstrated in Eq. (5.7), we can calculate $d_{z_2,1}$, $d_{z_2,2}$, $d_{z_3,2}$, $d_{z_3,3}$, $d_{z_1,3}$:

$$\begin{cases} d_{z_1,1} = H(g^{a_1 \cdot H(x_1||ID_z)}) \cdot d_{z_1} \mod q \\ d_{z_2,1} = H(g^{b_1 \cdot H(x_1||ID_z)}) \cdot d_{z_2} \mod q \\ d_{z_2,2} = H(g^{a_2 \cdot H(x_2||ID_z)}) \cdot d_{z_2} \mod q \\ d_{z_3,2} = H(g^{b_2 \cdot H(x_2||ID_z)}) \cdot d_{z_3} \mod q \\ d_{z_3,3} = H(g^{a_3 \cdot H(x_3||ID_z)}) \cdot d_{z_3} \mod q \\ d_{z_1,3} = H(g^{b_3 \cdot H(x_3||ID_z)}) \cdot d_{z_1} \mod q \end{cases} \tag{5.7}$$

Then, only HS_1 can decrypt $d_{z_1,1}, d_{z_2,1}$ for d_{z_1}, d_{z_2}; only HS_2 can decrypt $d_{z_2,2}, d_{z_3,2}$ for d_{z_2}, d_{z_3}; and only HS_3 can decrypt $d_{z_3,3}, d_{z_1,3}$ for d_{z_3}, d_{z_1}.

Commitment on the correction of d_z. In addition to keep the value d_z of invoice Z secret, we also need a mechanism for patient P to verify whether the published value of the invoice is correct, in other words, to make sure that $d_z = d_{z_1} + d_{z_2} + d_{z_3} \mod q$ are correct. To achieve this, we make use of the secret key S_z which is shared between P and the owner hospital HS_0 when the invoice Z was generated. That is to say, both P and HS_0 can use S_z to calculate the hash value $H_z = H(ID_z||S_z)$. Specifically, HS_0 first uses H_z to encrypt $d_{z_1}, d_{z_2}, d_{z_3}$ to generate $H_{z_1}, H_{z_2}, H_{z_3}$ and $g^{H_{z_1}}, g^{H_{z_2}}, g^{H_{z_3}}$:

$$\begin{cases} g^{H_{z_1}} \longleftarrow H_{z_1} = H_z \cdot d_{z_1} \mod q \\ g^{H_{z_2}} \longleftarrow H_{z_2} = H_z \cdot d_{z_2} \mod q \\ g^{H_{z_3}} \longleftarrow H_{z_3} = H_z \cdot d_{z_3} \mod q \end{cases} \tag{5.8}$$

Then, $H_{z_1}, H_{z_2}, H_{z_3}$ will be used to calculate the following commitments:

$$\begin{cases} C_1 = g^{(a_1+b_1+H_{z_1})^{-1}} \\ C_2 = g^{(a_2+b_2+H_{z_2})^{-1}} \\ C_3 = g^{(a_3+b_3+H_{z_3})^{-1}} \end{cases} \tag{5.9}$$

Note that, $(H_{z_1}, H_{z_2}, H_{z_3})$ will not be released, only $(g^{H_{z_1}}, g^{H_{z_2}}, g^{H_{z_3}})$ and the commitment (C_1, C_2, C_3) will be published. Then, with this commitment scheme, everybody can verify the values of $(H_{z_1}, H_{z_2}, H_{z_3})$. Take H_{z_1} for example, we can check if $e(C_1, g^{a_1} \cdot g^{b_1} \cdot g^{H_{z_1}}) \stackrel{?}{=} e(g, g)$, since

$$\begin{aligned} e(C_1, g^{a_1} \cdot g^{b_1} \cdot g^{H_{z_1}}) &= e(g^{(a_1+b_1+H_{z_1})^{-1}}, g^{a_1+b_1+H_{z_1}}) \\ &= e(g,g)^{(a_1+b_1+H_{z_1})^{-1} \cdot (a_1+b_1+H_{z_1})} = e(g,g) \end{aligned} \tag{5.10}$$

This commitment can be regarded as a variant of Zhang et al.'s short signature scheme [64], which is also *computationally binding* and *computation-*

ally hiding under the discrete logarithm assumption. Given $g^{(a_1+b_1+H_{z_1})^{-1}}$, $g^{(a_2+b_2+H_{z_2})^{-1}}$, $g^{(a_3+b_3+H_{z_3})^{-1}}$, it's hard for an attacker to calculate $H_{z_1}, H_{z_2}, H_{z_3}$ due to the discrete logarithm assumption. That makes the commitment *computationally hiding*. On the other hand, since $g^{a_1}, g^{b_1}, g^{a_2}, g^{b_2}, g^{a_3}, g^{b_3}$ have already been submitted on the blockchain with $\mathbb{D}_{init}$, it is also hard for an attacker to find $H'_{z_1}, H'_{z_2}, H'_{z_3}$ to satisfy $C_1 = g^{(a_1+b_1+H'_{z_1})^{-1}}$, $C_2 = g^{(a_2+b_2+H'_{z_2})^{-1}}$ and $C_3 = g^{(a_3+b_3+H'_{z_3})^{-1}}$. That makes this commitment *computationally binding*. With that in mind, the patient P can use

$$g^{H_z \cdot d_z} = g^{H_{z_1}} \cdot g^{H_{z_2}} \cdot g^{H_{z_3}} \tag{5.11}$$

to verify the sum of d_{z_1}, d_{z_2} and d_{z_3}, because

$$\begin{aligned} g^{H_z \cdot d_z} &= g^{H_{z_1}} \cdot g^{H_{z_2}} \cdot g^{H_{z_3}} \\ &= g^{H_z \cdot d_{z_1}} \cdot g^{H_z \cdot d_{z_2}} \cdot g^{H_z \cdot d_{z_3}} \\ &= g^{H_z \cdot (d_{z_1}+d_{z_2}+d_{z_3})} = g^{H_z \cdot d_z} \end{aligned} \tag{5.12}$$

So to sum it up, to publish the value d_z of an invoice Z to the blockchain, the owner hospital HS_0 sends the data $\mathbb{D}_{invoice}$ which contains patient identifier ID_P, invoice identifier ID_Z and the following values:

$$\mathbb{D}_{invoice} = \begin{bmatrix} ID_P, ID_Z \\ g^{H_{z_1}}, g^{H_{z_2}}, g^{H_{z_3}} \\ C_1, C_2, C_3 \\ d_{z_1,1}, d_{z_2,1} \\ d_{z_2,2}, d_{z_3,2} \\ d_{z_3,3}, d_{z_1,3} \end{bmatrix}$$

5.4 Patient Data Verification

From Eq. (5.11), patient P can verify if the published invoice value d_z is correct or not by calculating $H_z = H(ID_z||S_z)$, where ID_z is the invoice identifier and S_z is the shared secret between P and the owner hospital HS_0 when the invoice was issued. Specifically, patient P verifies the values $H_{z_1}, H_{z_2}, H_{z_3}$ by checking if:

$$\begin{cases} e(C_1, g^{a_1} \cdot g^{b_1} \cdot g^{H_{z_1}}) = e(g, g) \\ e(C_2, g^{a_2} \cdot g^{b_2} \cdot g^{H_{z_2}}) = e(g, g) \\ e(C_3, g^{a_3} \cdot g^{b_3} \cdot g^{H_{z_3}}) = e(g, g) \end{cases} \tag{5.13}$$

and then, from Eq. (5.11), P verifies the values of d_{z_1}, d_{z_2} and d_{z_3} with the equation $g^{H_z \cdot d_z} \stackrel{?}{=} g^{H_{z_1}} \cdot g^{H_{z_2}} \cdot g^{H_{z_3}}$. Because S_z is only known to patient

P, other people cannot calculate H_z and derive the real value of d_z from $g^{H_{z_1}}, g^{H_{z_2}}, g^{H_{z_3}}$.

5.5 Querying for Information

The insurance company I can send query transactions to the blockchain indicating its interest in learning the spending of patient P. To protect patient P's privacy, the insurance company I can only query for the total amount of money patient P has spent in the owner hospital HS_0. The query data $\mathbb{D}_{query}$ can simply be the patient identifier and hospital identifier, i.e., $\mathbb{D}_{query} = [ID_P, ID_{HS_0}]$. Upon receiving $\mathbb{D}_{query}$, the helper hospitals HS_1, HS_2, HS_3 can work to recover the spending of P and publish their responses to the blockchain, as described in the next subsection.

5.6 Generating Responses

Assuming patient P has k invoices in hospital HS_0, for each invoice Z, there is a piece of data $\mathbb{D}_{invoice}$ on the blockchain which contains the encrypted information of the invoice value d_z. To get the total amount of money that the patient P has spent in hospital HS_0, we need to get the sum

$$D = \sum_{z=1}^{k} d_z = \sum_{z=1}^{k} d_z \bmod q \tag{5.14}$$

Note that, it is reasonable to assume that $\sum_{z=1}^{k} d_z < q$ in real scenarios because q is large prime number, we have $\sum_{z=1}^{k} d_z = \sum_{z=1}^{k} d_z \bmod q$ in the above equation. Since we need to split d_z into $(d_{z_1}, d_{z_2}, d_{z_3})$, i.e., $d_z = d_{z_1} + d_{z_2} + d_{z_3} \bmod q$, to publish on the blockchain, we need to find

$$D = D_1 + D_2 + D_3 \bmod q \tag{5.15}$$

where

$$\begin{cases} D_1 = \sum_{z=1}^{k} d_{z_1} \bmod q \\ D_2 = \sum_{z=1}^{k} d_{z_2} \bmod q \\ D_3 = \sum_{z=1}^{k} d_{z_3} \bmod q \end{cases} \tag{5.16}$$

From Table 5.2 and Eq. (5.6), we have discussed that the helper hospital HS_1 can recover d_{z_1}, d_{z_2} from $d_{z_1,1}, d_{z_2,1}$ with the equations:

$$\begin{cases} d_{z_1} = d_{z_1,1} \cdot \dfrac{1}{H(g^{a_1 \cdot H(x_1 || ID_z)})} \mod q \\ d_{z_2} = d_{z_2,1} \cdot \dfrac{1}{H(g^{b_1 \cdot H(x_1 || ID_z)})} \mod q \end{cases} \tag{5.17}$$

Using the above equation, HS_1 can gather all d_{z_1}, d_{z_2} published on the blockchain with $1 \leq z \leq k$, and subsequently recover $D_1 = \sum_{z=1}^{k} d_{z_1} \mod q, D_2 = \sum_{z=1}^{k} d_{z_2} \mod q$. Similarly, HS_2 can recover D_2, D_3, and HS_3 can recover

D_3, D_1. It is easy to see that with the collaboration of any two helpers, the value D can be recovered.

The helpers then encrypt the sums they have calculated with insurance company I's public key:

$$\begin{cases} E_{12} = Encrypt_{pk_I}(D_1||D_2) \\ E_{23} = Encrypt_{pk_I}(D_2||D_3) \\ E_{31} = Encrypt_{pk_I}(D_3||D_1) \end{cases} \tag{5.18}$$

Finally, the helper hospital HS_1 respond to insurance company I's query by publishing $\mathbb{D}_{res_1} = [E_{12}]$, while HS_2 publishes $\mathbb{D}_{res_2} = [E_{23}]$, and HS_3 publishes $\mathbb{D}_{res_3} = [E_{31}]$ to the blockchain.

5.7 Gathering Responses

The insurance company I can recover the total amount of money D that patient P has spent by aggregating the answers from helper hospitals to get D_1, D_2, D_3. For example, in order to get D_1 and D_2, the insurance company I needs to decrypt the response from HS_1 with the private key:

$$Decrypt_{sk_I}(E_{12}) = Decrypt_{sk_I}(Encrypt_{pk_I}(D_1||D_2)) = (D_1||D_2) \tag{5.19}$$

Similarly, $(D_2||D_3)$ and $(D_3||D_1)$ can be acquired by decrypting the responses from HS_2 and HS_3 respectively. Note that, we only need the responses from any two hospitals to get all D_1, D_2 and D_3. Finally, the sum D can be obtained by calculating $D = D_1 + D_2 + D_3 \bmod q$. The correctness is as follows,

$$\begin{aligned}
D &= D_1 + D_2 + D_3 \bmod q \\
&= \left(\sum_{z=1}^{k} d_{z_1} \bmod q + \sum_{z=1}^{k} d_{z_2} \bmod q + \sum_{z=1}^{k} d_{z_3} \bmod q\right) \bmod q \\
&= \sum_{z=1}^{k} (d_{z_1} + d_{z_2} + d_{z_3}) \bmod q \\
&= \sum_{z=1}^{k} d_z \bmod q = \sum_{z=1}^{k} d_z \qquad \square
\end{aligned} \tag{5.20}$$

Chapter 6

Security Analysis and Performance Evaluation

In this chapter, we will make some security analysis and performance evaluation of our proposed blockchain-based privacy-preserving medical insurance storage system.

6.1 Security Analysis

First, we analyze the security of the proposed medical insurance storage system. Since it is based on the public blockchain, our system is *decentralized* and *tamper-resistant*. We achieve the *confidentiality* by using the secret keys (x_1, x_2, x_3), and use the $(2, 3)$-threshold secret sharing scheme to ensure the *high privacy and availability*, and also apply the commitment schemes to guarantee all published data are *verifiable*.

- **Decentralization and Tamper-resistance**: Our system is blockchain-based, so we inherit all the security properties of the public blockchain. The core of blockchain is *decentralization*. In our implementation, the data are stored on the Ethereum public blockchain, which is comprised of thousands of nodes. This makes the data extremely resilient even when the majority of the network is down, as there is no single point of failure. For critical data such as medical insurance in our system, this property is very important, as the patient can expect to never lose their data. Furthermore, blockchain technology also offers *tamper-resistance*. The "Proof-of-Work" consensus mechanism makes it impossible to rewrite the content of the blockchain without possessing at least 51% of the total computing power of the network. That means once a piece of data has been published on the blockchain, it's impossible for an attacker modify it. In our system, this means that when the owner hospital has submitted some data and that data has been verified, users can be sure that the data will always be correct.

  On the other hand, since system is blockchain-based, our data could be affected if an adversary launches an attack on the blockchain. For example, in an 51% attack, the attacker can rewrite the content of the blockchain. If that happens and the blocks containing our data are rewritten, our data may be lost. The Ethereum blockchain platform is gradually moving to use the "Proof-of-Stake" consensus protocol to prevent this kind of attack from happening.

- **Secret Keys**: We achieve the confidentiality through the use of secret keys (x_1, x_2, x_3). Because all communication in our system goes through the blockchain, it's easy for anyone to retrieve all of our data. By distributing x_1, x_2, x_3 to helper hospitals HS_1, HS_2, HS_3 respectively, the owner hospital HS_0 makes sure that nobody can read the messages not meant for them. The most important data to be published is the invoice value d_z, which is split into three parts $d_{z_1}, d_{z_2}, d_{z_3}$. Note that $d_z = d_{z_1} + d_{z_2} + d_{z_3} \mod q$, the modular operation makes it possible to hide d_z by choosing large $d_{z_1}, d_{z_2}, d_{z_3}$. Using (x_1, x_2, x_3) to encrypt $(d_{z_1}, d_{z_2}, d_{z_3})$ into $(d_{z_1,1},\ d_{z_2,1},\ d_{z_2,2},\ d_{z_3,2},\ d_{z_3,3},\ d_{z_1,3})$, we ensure only HS_1 can decrypt $d_{z_1,1}, d_{z_2,1}$ for d_{z_1}, d_{z_2}, only HS_2 can decrypt $d_{z_2,2}, d_{z_3,2}$ for d_{z_2}, d_{z_3}, and only HS_3 can decrypt $d_{z_3,3}, d_{z_1,3}$ for d_{z_3}, d_{z_1}.

- **Threshold Secret Sharing Scheme**: We achieve *privacy preservation* and *high availability* through our (2,3)-threshold secret sharing scheme. The role of helper hospitals HS_1, HS_2, HS_3 is to process the requests of insurance company I and response with the correct data. But we also do not want the the helper hospitals to learn more about the patient's data than they need to, our threshold scheme ensures that each helper HS_1, HS_2, HS_3 only have access to 2/3 of patient P's data.

For any invoice Z that has the value $d_z = d_{z_1} + d_{z_2} + d_{z_3} \mod q$, HS_1 can retrieve d_{z_1}, d_{z_2} and thus calculate the sum $D_1 = \sum d_{z_1}, D_2 = \sum d_{z_2}$ for all invoices. Similarly, HS_2 can retrieve $D_2 = \sum d_{z_2}, D_3 = \sum d_{z_3}$ and HS_3 can retrieve $D_3 = \sum d_{z_3}, D_1 = \sum d_{z_1}$. It is the insurance company I's responsibility to aggregate D_1, D_2, D_3 to get the final answer $D = D_1 + D_2 + D_3 \mod q$. The privacy of patient P is protected because neither HS_1, HS_2, HS_3 have a complete view of the patient's data, unless there is collusion between two helpers. And insurance company I can only get access to the total amount of money P has spent. Note that in case there is only one invoice, the total value of invoices equal the value of that single invoice, so there is a leak of privacy which we will try to address in future research. Finally, the threshold scheme also provides high availability to our system: Even if 1 out of 3 helper hospitals goes down, the remaining 2 can still complete the protocol without any problem, as the insurance company I only need two responses from helpers to get the desired answer.

- **Commitments**: We use cryptographic commitments to make sure that all data published to the blockchain can be verified without leaking sensitive information. Our commitment schemes are *computationally binding* and *computationally hiding* under the discrete logarithm assumption. In the *Initialization* step, we generate the secret keys

(x_1, x_2, x_3) by choosing random $(a_1, a_2, a_3, b_1, b_2, b_3)$ such that:

$$\begin{aligned} a_1 b_1 &= a_2 + b_2 + x_1 \mod q \\ a_2 b_2 &= a_3 + b_3 + x_2 \mod q \\ a_3 b_3 &= a_1 + b_1 + x_3 \mod q \end{aligned} \tag{6.1}$$

And commit to them by publishing $(g^{a_1},\ g^{a_2},\ g^{a_3},\ g^{b_1},\ g^{b_2},\ g^{b_3},\ g^{x_1},\ g^{x_2},\ g^{x_3})$. Without solving the discrete logarithm problem, an attacker cannot guess x_1 from g^{x_1} , nor can he generate x_1', x_2', x_3' such that $x_1' \neq x_1, x_2' \neq x_2, x_3' \neq x_3$ and $e(g^{a_1}, g^{b_1}) = e(g^{a_2} \cdot g^{b_2} \cdot g^{x_1}, g)$, $e(g^{a_2}, g^{b_2}) = e(g^{a_3} \cdot g^{b_3} \cdot g^{x_2}, g)$, and $e(g^{a_3}, g^{b_3}) = e(g^{a_1} \cdot g^{b_1} \cdot g^{x_3}, g)$.

In the *Data Publication* step, to enable patient P to verify the value d_z of invoice Z, we make use of a secret value H_z which can only be calculated by patient P and owner hospital HS_0. We split d_z into $d_{z_1}, d_{z_2}, d_{z_3}$ with the equation $d_z = d_{z_1} + d_{z_2} + d_{z_3} \mod q$. Encrypting $d_{z_1}, d_{z_2}, d_{z_3}$ with H_z gives us $H_{z_1}, H_{z_2}, H_{z_3}$. We then commit to $H_{z_1}, H_{z_2}, H_{z_3}$ by generating $g^{H_{z_1}}, g^{H_{z_2}}, g^{H_{z_3}}$ and:

$$\begin{aligned} C_1 &= g^{(a_1+b_1+H_{z_1})^{-1}} \\ C_2 &= g^{(a_2+b_2+H_{z_2})^{-1}} \\ C_3 &= g^{(a_3+b_3+H_{z_3})^{-1}} \end{aligned}$$

Similar as the above, we can see that $(g^{H_{z_1}}, g^{H_{z_2}}, g^{H_{z_3}}, C_1, C_2, C_3)$ are computationally binding and computationally hiding under the discrete logarithm assumption. Note that we need to generate $g^{H_{z_1}}, g^{H_{z_2}}, g^{H_{z_3}}$

instead of just using $g^{d_{z1}}, g^{d_{z2}}, g^{d_{z3}}$ because an attacker can get $g^{d_z} = g^{d_{z1}} \cdot g^{d_{z2}} \cdot g^{d_{z3}}$. Since d_z is a small number, it's possible to guess d_z from g^{d_z} in that way. Lastly, patient P can confirm that $d_{z_1}, d_{z_2}, d_{z_3}$ are correct by checking if $g^{H_z \cdot d_z} \stackrel{?}{=} g^{H_{z_1}} \cdot g^{H_{z_2}} \cdot g^{H_{z_3}}$.

Based on the above analysis, we can see our proposed blockchain-based medical insurance storage system is indeed a secure, reliable, and privacy-preserving system due to the blockchain and other cryptographic techniques employed in the system.

6.2 Performance Evaluation

In this section we evaluate the performance of our proposed system. We begin by describing our system configuration and then take a look at the transaction size and processing time when running the system on the blockchain.

6.2.1 System Configuration

Because our system is blockchain-based, a large part of the performance is dependent on the blockchain platform. In this thesis, we use Ethereum as our blockchain platform, which is currently the most widely used platform for developing decentralized applications. We implement and run our system on a Macbook Pro with the following specs: Intel Core i7 - 4770 CPU with

2.2GHz, 16GB memory, Mac OS X 10.13.6. We develop our system using the Truffle suite (`https://truffleframework.com`), which is the most popular Ethereum Smart Contract development environment at the time of this writing. We then deploy a local Ethereum blockchain network with Ganache - the built-in private blockchain included in Truffle - which closely mimics the main Ethereum network. Ganache also provides a visual blockchain explorer to make it easier for developers to navigate the blockchain data. Figure 6.1 shows a screenshot of the Ganache blockchain explorer.

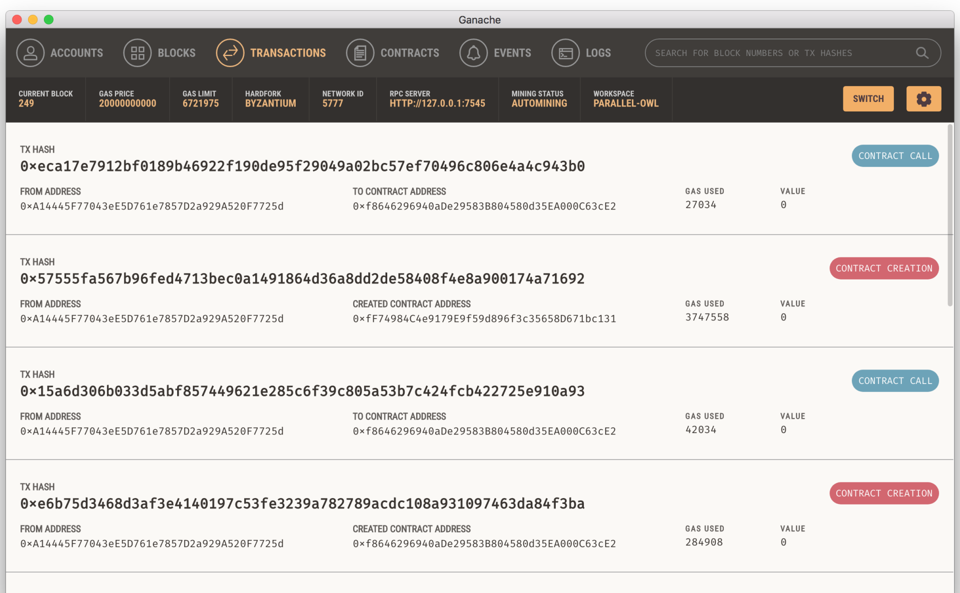

Figure 6.1: A screenshot of the Ganache blockchain explorer showing the list of transactions in the local blockchain

In our system, we use the ECDSA signature scheme with the secp256k1 elliptic curve [36] that is included in Ethereum. For encrypting messages such as secret keys and query responses, we an Elliptic Curve Integrated Encryption Scheme (ECIES) with AES on the secp256k1 curve where the message is encrypted with the receiver's Ethereum wallet public key and decrypted with the corresponding private key. Moreover, for our commmitment schemes, we make use of the 256-bit Barreto-Naehrig curve BN(2,254) [7]. The curve supports the asymmetric pairing operation $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are additive groups and $\mathbb{G}_T$ is a multiplicative group. To commit a secret s, we calculate the value sg_1 or sg_2, where g_1 is the generator of $\mathbb{G}_1$ and g_2 is the generator of $\mathbb{G}_2$. The size of a point in $\mathbb{G}_1$ is 258 bits, and the size of a point in $\mathbb{G}_2$ is 1019 bits, both of which exceed the storage size limit of 256 bits for an integer in an Ethereum Smart contract. For that reason, we need to encode the commitments as `string`, which has a considerable impact on performance.

6.2.2 Evaluation Results

Because our prototype system is based on Ethereum, we are also limited by the constraints of the blockchain platform. In particular, the maximum number of arguments that can be sent for each function. Ethereum has a fixed limit of 16 fixed-length local variables (including arguments) per function, and `string` variables take double space. That means we can only send at most 8 strings per transaction. For the *Initialization* transaction, since some commitments need to be computed for both g_1 and g_2, we need to send 18 variables, most of which are represented as `string`:

$$\mathbb{D}_{init} = \begin{bmatrix} g_1a_1, g_1a_2, g_1a_3 \\ g_1b_1, g_1b_2, g_1b_3 \\ g_2a_1, g_2a_2, g_2a_3 \\ g_2b_1, g_2b_2, g_2b_3 \\ g_1x_1, g_1x_2, g_1x_3 \\ E_{pk_1}(x_1), E_{pk_2}(x_2), E_{pk_3}(x_3) \end{bmatrix}$$

Which clearly exceeds the transaction variable limit, so we split $\mathbb{D}_{init}$ into three parts:

$$\mathbb{D}_{init}^1 = \begin{bmatrix} g_1a_1, g_1a_2, g_1a_3 \\ g_1b_1, g_1b_2, g_1b_3 \end{bmatrix}, \mathbb{D}_{init}^2 = \begin{bmatrix} g_2a_1, g_2a_2, g_2a_3 \\ g_2b_1, g_2b_2, g_2b_3 \end{bmatrix}$$

$$\mathbb{D}_{init}^3 = \begin{bmatrix} g_1x_1, g_1x_2, g_1x_3 \\ E_{pk_1}(x_1), E_{pk_2}(x_2), E_{pk_3}(x_3) \end{bmatrix}$$

And publish the parts separately in 3 different transactions. Similarly, the invoice data transaction $\mathbb{D}_{invoice}$ also needs to be split into two parts:

$$\mathbb{D}^1_{invoice} = \begin{bmatrix} ID_P, ID_Z \\ d_{z_1,1}, d_{z_2,1} \\ d_{z_2,2}, d_{z_3,2} \\ d_{z_3,3}, d_{z_1,3} \end{bmatrix}, \mathbb{D}^2_{invoice} = \begin{bmatrix} ID_P, ID_Z \\ g^{H_{z_1}}, g^{H_{z_2}}, g^{H_{z_3}} \\ C_1, C_2, C_3 \end{bmatrix}$$

The size for each transaction data in the prototype system is shown in Table 6.1

Data	Size
$\mathbb{D}^1_{init}$	964 bytes
$\mathbb{D}^2_{init}$	1924 bytes
$\mathbb{D}^3_{init}$	1731 bytes
$\mathbb{D}^1_{invoice}$	228 bytes
$\mathbb{D}^2_{invoice}$	1475 bytes
$\mathbb{D}_{query}$	67 bytes
$\mathbb{D}^1_{res}$	580 bytes
$\mathbb{D}^2_{res}$	580 bytes
$\mathbb{D}^3_{res}$	580 bytes

Table 6.1: Sizes of transactions used in our prototype system

Note that the transaction sizes are quite large, this is due to using the `string` data type for some variables to get around the 256-bit limit of `integer`. To reduce the transaction sizes, we may choose some elliptic curves with smaller field modulus so that the variables could fit into 256 bits, or split up a large number into smaller numbers that are less than 256 bits.

We will then look at the performance evaluation of the system. For each step of the system, we perform the action 100 times and calculate their average time cost to process the transactions. The average time for all steps are shown in Table 6.2

Step	Time Cost
Initialize	120 ms
Verify Initialization	866 ms
Data Publication	760 ms
Data Verification	812 ms
Query	6 ms
Respond To Query	833 ms
Response Collection	130 ms

Table 6.2: Average time cost to process transactions

It should be noted that the sixth step, *Respond To Query*, depends on the number of invoices published on the blockchain. The larger the number of invoices, the longer it takes to get the sum of parts. The time cost in Table 6.2 is calculated when there are 2 published invoices on the blockchain. Similar to the transaction size, the performance of the system can also be increased by using more efficient data types. Currently the performance is affected by the use of many `string` conversion operations.

Overall, the efficiency of our prototype system is dependent on the Ethereum blockchain. Even though there is no limit to the size of a transaction and a block, we are still constrained by the transaction cost, which gets higher the more data we send. Furthermore, the block interval of 15 seconds means that we cannot get very high throughput. If there is a more suitable blockchain platform, we may be able to get better performance out of the system. The source code for the prototype system can be found in Appendix A.

Chapter 7

Conclusions and Future Work

In this chapter, we summarize our contribution in this thesis, propose our future research work, and give our final remarks.

7.1 Conclusions

In this thesis, we proposed a blockchain-based privacy-preserving medical insurance storage system. With blockchain integration, our system takes advantage of decentralization, tamper resistance, privacy-preservation and high availability. Because of the blockchain's decentralization, parties can communicate without the need of a central entity, reducing the cost and eliminate the single point of failure. The data recorded in the system is highly credible due to the tamper-resistance property of blockchain. Furthermore, from the use of a threshold scheme to share secrets, sensitive patient data is confidential as long as the helper hospitals are honest, effectively preserving the privacy of patients. By using a (2,3)-threshold secret sharing scheme, the system is very fast to respond to requests with minimum computation. The use of multiple helper hospitals also ensure high availability for the system: even if a helper hospital fails, other helpers can still do their job to process data for insurance companies. Finally, a performance evaluation of the system is given.

7.2 Future Work

In our future work, we will support more types of data such as Electronic Health Records to create a complete medical storage ecosystem. The challenge lies in discovering a better scheme to reduce the memory footprint for data stored on the blockchain and improve security measures. We will also try to build our system on Hyperledger to take advantage of the permissioned blockchain, with out-of-the-box implementation for access control, privacy, high scalability and high performance.

Bibliography

[1] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al., *Hyperledger fabric: a distributed operating system for permissioned blockchains*, Proceedings of the Thirteenth EuroSys Conference, ACM, 2018, p. 30.

[2] Andreas M. Antonopoulos, *Mastering bitcoin: Unlocking digital cryptocurrencies*, 1st ed., O'Reilly Media, Inc., 2014.

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

[4] Microsoft Azure, *Microsoft azure: Blockchain as a service*, Accessed: 2019-01-28.

[5] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille, *Enabling blockchain innovations with pegged sidechains*, URL: http://www. opensciencereview. com/papers/123/enablingblockchain-innovations-with-pegged-sidechains (2014).

[6] Tobias Bamert, Christian Decker, Roger Wattenhofer, and Samuel Welten, *Bluewallet: The secure bitcoin wallet*, International Workshop on Security and Trust Management, Springer, 2014, pp. 65–80.

[7] Paulo SLM Barreto and Michael Naehrig, *Pairing-friendly elliptic curves of prime order*, International Workshop on Selected Areas in Cryptography, Springer, 2005, pp. 319–331.

[8] Imran Bashir, *Mastering blockchain*, Packt Publishing Ltd, 2017.

[9] Alireza Beikverdi and JooSeok Song, *Trend of centralization in bitcoin's distributed network*, 2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), IEEE, 2015, pp. 1–6.

[10] Mihir Bellare and Phillip Rogaway, *Random oracles are practical: A paradigm for designing efficient protocols*, CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993., 1993, pp. 62–73.

[11] Giancarlo Bigi, Andrea Bracciali, Giovanni Meacci, and Emilio Tuosto, *Validation of decentralised smart contracts through game theory and formal methods*, Programming Languages with Applications to Biology and Security, Springer, 2015, pp. 142–161.

[12] Bithalo.org, *Bithalo documentation*, `https://bithalo.org/Documentation.doc`, Accessed: 2019-02-09.

[13] Dan Boneh and Xavier Boyen, *Short signatures without random oracles*, International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2004, pp. 56–73.

[14] Dan Boneh and Matt Franklin, *Identity-based encryption from the weil pairing*, Annual international cryptology conference, Springer, 2001, pp. 213–229.

[15] Dan Boneh, Ben Lynn, and Hovav Shacham, *Short signatures from the weil pairing*, International Conference on the Theory and Application of Cryptology and Information Security, Springer, 2001, pp. 514–532.

[16] BTCManager.com, *'51 percent attack' hits krypton, moves to pos*, `https://btcmanager.com/51-percent-attack-hits-krypton-moves-to-pos/`, Accessed: 2019-02-09.

[17] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell, *Bulletproofs: Short proofs for confidential*

transactions and more, 2018 IEEE Symposium on Security and Privacy (SP), IEEE, 2018, pp. 315–334.

[18] Vitalik Buterin, *Ethereum: A next-generation smart contract and decentralized application platform*, 2014, Accessed: 2018-10-22.

[19] Vitalik Buterin and Virgil Griffith, *Casper the friendly finality gadget*, arXiv preprint arXiv:1710.09437 (2017).

[20] Marco Conoscenti, Antonio Vetro, and Juan Carlos De Martin, *Blockchain for the internet of things: A systematic literature review*, Computer Systems and Applications (AICCSA), 2016 IEEE/ACS 13th International Conference of, IEEE, 2016, pp. 1–6.

[21] Christian Decker and Roger Wattenhofer, *Bitcoin transaction malleability and mtgox*, Computer Security - ESORICS 2014 (Cham) (Mirosław Kutyłowski and Jaideep Vaidya, eds.), Springer International Publishing, 2014, pp. 313–326.

[22] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel, *Ripemd-160: A strengthened version of ripemd*, International Workshop on Fast Software Encryption, Springer, 1996, pp. 71–82.

[23] Alevtina Dubovitskaya, Zhigang Xu, Samuel Ryu, Michael Schumacher, and Fusheng Wang, *Secure and trustable electronic medical records sharing using blockchain*, AMIA Annual Symposium Proceedings, vol. 2017, American Medical Informatics Association, 2017, p. 650.

[24] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse, *Bitcoin-ng: A scalable blockchain protocol*, 13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16), 2016, pp. 45–59.

[25] Ittay Eyal and Emin Gün Sirer, *Majority is not enough: Bitcoin mining is vulnerable*, Communications of the ACM **61** (2018), no. 7, 95–102.

[26] Sebastian Feld, Mirco Schönfeld, and Martin Werner, *Analyzing the deployment of bitcoin's p2p network under an as-level perspective*, Procedia Computer Science **32** (2014), 1121–1126.

[27] Fortune.com, *Bitcoin spinoff hacked in rare '51% attack'*, `http://fortune.com/2018/05/29/bitcoin-gold-hack/`, Accessed: 2019-02-09.

[28] Ghash.io, *51% discussion*, `https://assets.documentcloud.org/documents/1219875/51-discussion.pdf`, Accessed: 2019-02-09.

[29] Laura Gonzalez, *Blockchain, herding and trust in peer-to-peer lending*, Forthcoming, Managerial Finance Special Issue on Blockchain (2018).

[30] William J Gordon and Christian Catalini, *Blockchain technology for healthcare: facilitating the transition to patient-driven interoperability*, Computational and structural biotechnology journal **16** (2018), 224–230.

[31] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg, *Tumblebit: An untrusted bitcoin-compatible*

anonymous payment hub, Network and Distributed System Security Symposium, 2017.

[32] Ethan Heilman, Alison Kendler, and Aviv Zohar, *Eclipse attacks on bitcoin's peer-to-peer network.*

[33] Health Level Seven International (HL7), *Introducing hl7 fhir n.d*, Accessed: 2018-10-22.

[34] IBM, *Ibm blockchain*, Accessed: 2019-01-28.

[35] Drew Ivan, *Moving toward a blockchain-based method for the secure storage of patient records*, 2016.

[36] Don Johnson, Alfred Menezes, and Scott Vanstone, *The elliptic curve digital signature algorithm (ecdsa)*, International journal of information security **1** (2001), no. 1, 36–63.

[37] Divya Kamboj and T Andrew Yang, *An exploratory analysis of blockchain: Applications, security, and related issues*, Proceedings of the International Conference on Scientific Computing (CSC), The Steering Committee of The World Congress in Computer Science, Computer âĂę, 2018, pp. 67–73.

[38] Record Keeper, *Supply chain management over the recordskeeper blockchain*, Accessed: 2019-01-28.

[39] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov, *Ouroboros: A provably secure proof-of-stake blockchain protocol*, Annual International Cryptology Conference, Springer, 2017, pp. 357–388.

[40] Junichi Kishigami, Shigeru Fujimura, Hiroki Watanabe, Atsushi Nakadaira, and Akihiko Akutsu, *The blockchain-based digital content distribution system*, Big Data and Cloud Computing (BDCloud), 2015 IEEE Fifth International Conference on, IEEE, 2015, pp. 187–190.

[41] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou, *Hawk: The blockchain model of cryptography and privacy-preserving smart contracts*, 2016 IEEE symposium on security and privacy (SP), IEEE, 2016, pp. 839–858.

[42] Philip Koshy, Diana Koshy, and Patrick McDaniel, *An analysis of anonymity in bitcoin using p2p network traffic*, International Conference on Financial Cryptography and Data Security, Springer, 2014, pp. 469–485.

[43] Alexander Kuzmin, *Blockchain-based structures for a secure and operate iot*, Internet of Things Business Models, Users, and Networks, 2017, IEEE, 2017, pp. 1–7.

[44] Xiaoqi Li, Peng Jiang, Ting Chen, Xiapu Luo, and Qiaoyan Wen, *A survey on the security of blockchain systems*, Future Generation Computer Systems (2017).

[45] Rongxing Lu, *Security and privacy preservation in vehicular social networks*, (2012).

[46] Sihua Ma et al., *Using blockchain to build decentralized access control in a peer-to-peer e-learning platform*, Ph.D. thesis, University of Saskatchewan, 2018.

[47] Medicalchain, *Medicalchain white paper*, Accessed: 2018-10-28.

[48] Sarah Meiklejohn and Claudio Orlandi, *Privacy-enhancing overlays in bitcoin*, International Conference on Financial Cryptography and Data Security, Springer, 2015, pp. 127–141.

[49] Massimo Morini, *From'blockchain hype'to a real business case for financial markets*, Available at SSRN 2760184 (2016).

[50] Satoshi Nakamoto, *Bitcoin: A peer-to-peer electronic cash system.*

[51] Torben Pryds Pedersen, *Non-interactive and information-theoretic secure verifiable secret sharing*, Annual International Cryptology Conference, Springer, 1991, pp. 129–140.

[52] Gareth W Peters and Efstathios Panayi, *Understanding modern banking ledgers through blockchain technologies: Future of transaction processing*

and smart contracts on the internet of money, Banking beyond banks and money, Springer, 2016, pp. 239–278.

[53] Joseph Poon and Thaddeus Dryja, *The bitcoin lightning network: Scalable off-chain instant payments*, 2016.

[54] Silas Richelson, *Cryptographic protocols with strong security: Non-malleable commitments, concurrent zero-knowledge and topology-hiding multi-party computation*, Ph.D. thesis, UCLA, 2014.

[55] Adi Shamir, *How to share a secret*, Communications of the ACM **22** (1979), no. 11, 612–613.

[56] Nick Szabo, *Smart contracts*, Unpublished manuscript (1994).

[57] Florian Tschorsch and Björn Scheuermann, *Bitcoin and beyond: A technical survey on decentralized digital currencies*, IEEE Communications Surveys & Tutorials **18**, no. 3, 2084–2123.

[58] John Turek and Dennis Shasha, *The many faces of consensus in distributed systems*, Computer **25** (1992), no. 6, 8–17.

[59] Marie Vasek, Micah Thornton, and Tyler Moore, *Empirical analysis of denial-of-service attacks in the bitcoin ecosystem*, International Conference on Financial Cryptography and Data Security, Springer, 2014, pp. 57–71.

[60] Zhiguo Wan, Robert H Deng, and David Lee, *Electronic contract signing without using trusted third party*, International Conference on Network and System Security, Springer, 2015, pp. 386–394.

[61] Qi Xia, Emmanuel Boateng Sifah, Kwame Omono Asamoah, Jianbin Gao, Xiaojiang Du, and Mohsen Guizani, *Medshare: Trust-less medical data sharing among cloud service providers via blockchain*, IEEE Access **5** (2017), 14757–14767.

[62] Jesse Yli-Huumo, Deokyoon Ko, Sujin Choi, Sooyong Park, and Kari Smolander, *Where is current research on blockchain technology?âĂŤa systematic review*, PloS one **11** (2016), no. 10, e0163477.

[63] Xiao Yue, Huiju Wang, Dawei Jin, Mingqiang Li, and Wei Jiang, *Healthcare data gateways: found healthcare intelligence on blockchain with novel privacy risk control*, Journal of medical systems **40** (2016), no. 10, 218.

[64] Fangguo Zhang, Reihaneh Safavi-Naini, and Willy Susilo, *An efficient signature scheme from bilinear pairings and its applications*, International Workshop on Public Key Cryptography, Springer, 2004, pp. 277–290.

[65] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang, *Blockchain challenges and opportunities: a survey*, International Journal of Web and Grid Services **14** (2018), no. 4, 352–375.

[66] Lijing Zhou, Licheng Wang, and Yiru Sun, *Mistore: a blockchain-based medical insurance storage system*, Journal of medical systems **42** (2018), no. 8, 149.

[67] Guy Zyskind, Oz Nathan, et al., *Decentralizing privacy: Using blockchain to protect personal data*, Security and Privacy Workshops (SPW), 2015 IEEE, IEEE, 2015, pp. 180–184.

Appendix A

Source Code

A.1 Smart Contract in Solidity

```
pragma solidity >=0.5.0;


contract Bmis {
    struct InitializationData {
        string g1a1; string g1a2; string g1a3;
        string g1b1; string g1b2; string g1b3;
    }


    struct InitializationData2 {
        string g2a1; string g2a2; string g2a3;
        string g2b1; string g2b2; string g2b3;
    }

```

```
struct InitializationData3 {
    string g1x1; string g1x2; string g1x3;
    string ex1; string ex2; string ex3;
}

InitializationData public initData;
InitializationData2 public initData2;
InitializationData3 public initData3;

uint public invoicesCount = 0;
struct InvoiceData {
    uint invoiceID;
    uint patientID;
    uint256 dz11; uint256 dz21;
    uint256 dz22; uint256 dz32;
    uint256 dz33; uint256 dz13;
}

struct InvoiceData2 {
    uint invoiceID;
    uint patientID;
    string ghz1; string ghz2; string ghz3;
    string c1; string c2; string c3;
}

mapping (uint => InvoiceData) public invoices1;
mapping (uint => InvoiceData2) public invoices2;

uint public requestsCount = 0;
```

```
struct DataRequest {
    uint requestID;
    address senderAddress;
    uint patientID;
    uint hospitalID;
    uint responseCount;
    mapping (uint => DataResponse) responses;
}
mapping (uint => DataRequest) public requests;

struct DataResponse {
    address senderAddress;
    string e;
}

function initialize (string memory g1a1, string memory g1a2, string memory
    ↪ g1a3,
    string memory g1b1, string memory g1b2, string memory g1b3)
    public {
        initData = InitializationData(g1a1, g1a2, g1a3, g1b1, g1b2, g1b3);
}

function initialize2 (string memory g2a1, string memory g2a2, string memory
    ↪  g2a3,
    string memory g2b1, string memory g2b2, string memory g2b3) public {
        initData2 = InitializationData2(g2a1, g2a2, g2a3, g2b1, g2b2, g2b3)
            ↪ ;
}
```

```
function initialize3 (string memory g1x1, string memory g1x2, string memory
    ↪ g1x3,
    string memory ex1, string memory ex2, string memory ex3) public {
        initData3 = InitializationData3(g1x1, g1x2, g1x3, ex1, ex2, ex3);
}

function publishData (
    uint256 patientID,
    uint256 dz11, uint256 dz21,
    uint256 dz22, uint256 dz32,
    uint256 dz33, uint256 dz13) public {
        invoices1[invoicesCount + 1] = InvoiceData(invoicesCount + 1,
            ↪ patientID, dz11, dz21, dz22, dz32, dz33, dz13);
}

function publishData2 (
    uint256 patientID,
    string memory ghz1, string memory ghz2, string memory ghz3,
    string memory c1, string memory c2, string memory c3) public {
        invoices2[invoicesCount + 1] = InvoiceData2(invoicesCount + 1,
            ↪ patientID, ghz1, ghz2, ghz3, c1, c2, c3);
        invoicesCount++;
}

function requestData (uint256 patientID, uint256 hospitalID) public {
    address from = msg.sender;
    requestsCount++;
    requests[requestsCount] = DataRequest(requestsCount, from, patientID,
        ↪ hospitalID, 0);
```

```
    }

    function respond(uint requestID, string memory e) public {
        address from = msg.sender;
        DataRequest storage request = requests[requestID];
        request.responseCount = request.responseCount + 1;
        request.responses[request.responseCount] = DataResponse(from, e);
    }

    function viewResponse(uint requestID, uint responseID) public view returns
        ↪ (string memory response) {
        response = requests[requestID].responses[responseID].e;
    }
}
```

A.2 Client Code in Python

```
import time
import json
import helper
import time
from web3 import Web3, HTTPProvider

(G, o, g1, g2, e) = helper.parameters()

contract_address = '0xFE4D7D45ce3f437D8dBF111181FC7bdc1264AE08'
```

```
wallet_private_key = '0
    ↪ xb662d22ad7fc7989f0d6b6524f72d324119899f20fd8bd0bf707f51a691499b2'
wallet_address = '0xA14445F77043eE5D761e7857D2a929A520F7725d'

my_provider = HTTPProvider('http://127.0.0.1:7545')
w3 = Web3(my_provider)

compiledContractFile = open('../build/contracts/Bmis.json')
compiledContract = json.load(compiledContractFile)

app = w3.eth.contract(address = contract_address, abi = compiledContract['abi'
    ↪ ])

ownerKey = helper.getKeyPair('
    ↪ b662d22ad7fc7989f0d6b6524f72d324119899f20fd8bd0bf707f51a691499b2')
helper1Key = helper.getKeyPair('8701
    ↪ f58d2010c81089ab1f3a254b0526763ffad1ba5ece25e0b91a4f23e9eb14')
helper2Key = helper.getKeyPair('
    ↪ e64ceef450a72032d509c453abe52814590010c100b27c899467638800aa4cd1')
helper3Key = helper.getKeyPair('0
    ↪ a2fe1a7ec603c6baaef54d92425b134027420dae2de6b116a13a2c8339326b4')
insuranceKey = helper.getKeyPair('9
    ↪ a3ec09ef357903cf1ec5b4bb441b412e61446283e8229aa1b691124fc6aed5e')

def initialize():
    w3.eth.defaultAccount = w3.eth.accounts[0]

    (a1, a2, a3, b1, b2, b3) = (o.random(), o.random(), o.random(), o.random(),
        ↪  o.random(), o.random())
```

```
x1 = (a1*b1 - (a2 + b2)) % o
x2 = (a2*b2 - (a3 + b3)) % o
x3 = (a3*b3 - (a1 + b1)) % o

# save initialization data
f = open('initialization.txt', 'w')
f.write(hex(a1)), f.write('\n')
f.write(hex(a2)), f.write('\n')
f.write(hex(a3)), f.write('\n')
f.write(hex(b1)), f.write('\n')
f.write(hex(b2)), f.write('\n')
f.write(hex(b3)), f.write('\n')
f.write(hex(x1)), f.write('\n')
f.write(hex(x2)), f.write('\n')
f.write(hex(x3)), f.write('\n')
f.close()

g1a1, g1a2, g1a3, g1b1, g1b2, g1b3 = g1*a1, g1*a2, g1*a3, g1*b1, g1*b2, g1*
    ↪ b3
g2a1, g2a2, g2a3, g2b1, g2b2, g2b3 = g2*a1, g2*a2, g2*a3, g2*b1, g2*b2, g2*
    ↪ b3
g1x1, g1x2, g1x3, g2x1, g2x2, g2x3 = g1*x1, g1*x2, g1*x3, g2*x1, g2*x2, g2*
    ↪ x3
ex1 = helper.encryptData(helper1Key.public_key.to_hex(), hex(x1))
ex2 = helper.encryptData(helper2Key.public_key.to_hex(), hex(x2))
ex3 = helper.encryptData(helper3Key.public_key.to_hex(), hex(x3))

app.functions.initialize(g1a1.export().hex(), g1a2.export().hex(), g1a3.
    ↪ export().hex(), g1b1.export().hex(), g1b2.export().hex(), g1b3.
```

```
        ↪ export().hex()).transact()
    app.functions.initialize2(g2a1.export().hex(), g2a2.export().hex(), g2a3.
        ↪ export().hex(), g2b1.export().hex(), g2b2.export().hex(), g2b3.
        ↪ export().hex()).transact()
    app.functions.initialize3(g1x1.export().hex(), g1x2.export().hex(), g1x3.
        ↪ export().hex(), ex1.hex(), ex2.hex(), ex3.hex()).transact()



def verifyInitialization():
    [hg1x1, hg1x2, hg1x3, hex1, hex2, hex3] = app.functions.initData3().call()

    [g1a1, g1a2, g1a3, g1b1, g1b2, g1b3] = [helper.strToG1Point(x) for x in app
        ↪ .functions.initData().call()]
    [g2a1, g2a2, g2a3, g2b1, g2b2, g2b3] = [helper.strToG2Point(x) for x in app
        ↪ .functions.initData2().call()]
    [g1x1, g1x2, g1x3] = [helper.strToG1Point(x) for x in [hg1x1, hg1x2, hg1x3
        ↪ ]]

    assert G.pair(g1a1,g2b1) == G.pair(g1a2 + g1b2 + g1x1, g2)
    assert G.pair(g1a2,g2b2) == G.pair(g1a3 + g1b3 + g1x2, g2)
    assert G.pair(g1a3,g2b3) == G.pair(g1a1 + g1b1 + g1x3, g2)

    x1 = helper.decryptData(helper1Key.to_hex(), bytes.fromhex(hex1))
    x2 = helper.decryptData(helper2Key.to_hex(), bytes.fromhex(hex2))
    x3 = helper.decryptData(helper3Key.to_hex(), bytes.fromhex(hex3))

    # save the secret keys
    f = open('keys.txt', 'w')
    f.write(x1), f.write('\n')
```

```
    f.write(x2), f.write('\n')
    f.write(x3), f.write('\n')
    f.close()

def publishData():
    w3.eth.defaultAccount = w3.eth.accounts[0]

    [a1, a2, a3, b1, b2, b3, x1, x2, x3] = helper.getKeysFromTextFile('
        ↪ initialization.txt')
    [g1a1, g1a2, g1a3, g1b1, g1b2, g1b3] = [helper.strToG1Point(x) for x in app
        ↪ .functions.initData().call()]

    invoicesCount = app.functions.invoicesCount().call()
    IDz = invoicesCount + 1
    IDp = 1
    Sz = o.random()

    dz = helper.generateInvoiceValue()
    dz1 = o.random()
    dz2 = o.random()
    dz3 = (dz - dz1 - dz2) % o

    Hx1z = helper.bnToHash(x1 + IDz)
    Hx2z = helper.bnToHash(x2 + IDz)
    Hx3z = helper.bnToHash(x3 + IDz)

    dz11 = (helper.pointToIntHash(g1a1 * Hx1z) * dz1) % o
    dz21 = (helper.pointToIntHash(g1b1 * Hx1z) * dz2) % o
    dz22 = (helper.pointToIntHash(g1a2 * Hx2z) * dz2) % o
```

```
dz32 = (helper.pointToIntHash(g1b2 * Hx2z) * dz3) % o
dz33 = (helper.pointToIntHash(g1a3 * Hx3z) * dz3) % o
dz13 = (helper.pointToIntHash(g1b3 * Hx3z) * dz1) % o

Hz = helper.bnToHash(IDz + Sz)
Hz1 = (Hz * dz1) % o
Hz2 = (Hz * dz2) % o
Hz3 = (Hz * dz3) % o

g2hz1 = g2*Hz1
g2hz2 = g2*Hz2
g2hz3 = g2*Hz3

a1b1hz1_inv = (a1 + b1 + Hz1).mod_inverse(o)
a2b2hz2_inv = (a2 + b2 + Hz2).mod_inverse(o)
a3b3hz3_inv = (a3 + b3 + Hz3).mod_inverse(o)

C1 = g1 * a1b1hz1_inv
C2 = g1 * a2b2hz2_inv
C3 = g1 * a3b3hz3_inv

f = open('invoice' + str(IDz) + '.txt', 'w')
f.write(str(hex(dz))), f.write('\n')
f.write(str(hex(dz1))), f.write('\n')
f.write(str(hex(dz2))), f.write('\n')
f.write(str(hex(dz3))), f.write('\n')
f.write(str(hex(Sz))), f.write('\n')

app.functions.publishData(IDp, dz11.int(), dz21.int(), dz22.int(), dz32.int
```

```
        ↪ (), dz33.int(), dz13.int()).transact()
    app.functions.publishData2(IDp, g2hz1.export().hex(), g2hz2.export().hex(),
        ↪  g2hz3.export().hex(), C1.export().hex(), C2.export().hex(), C3.
        ↪ export().hex()).transact()



def invoiceVerification(IDz):
    # Everyone can check this part
    [IDz, IDp, dz11, dz21, dz22, dz32, dz33, dz13] = app.functions.invoices1(
        ↪ IDz).call()
    [IDz, IDp, hg2hz1, hg2hz2, hg2hz3, hC1, hC2, hC3] = app.functions.invoices2
        ↪ (IDz).call()

    [g2a1, g2a2, g2a3, g2b1, g2b2, g2b3] = [helper.strToG2Point(x) for x in app
        ↪ .functions.initData2().call()]
    [g2hz1, g2hz2, g2hz3] = [helper.strToG2Point(x) for x in [hg2hz1, hg2hz2,
        ↪ hg2hz3]]
    [C1, C2, C3] = [helper.strToG1Point(x) for x in [hC1, hC2, hC3]]

    assert G.pair(C1, g2a1 + g2b1 + g2hz1) == G.pair(g1, g2)
    assert G.pair(C2, g2a2 + g2b2 + g2hz2) == G.pair(g1, g2)
    assert G.pair(C3, g2a3 + g2b3 + g2hz3) == G.pair(g1, g2)

    # Only patient can check this part
    [dz, dz1, dz2, dz3, Sz] = helper.getKeysFromTextFile('invoice' + str(IDz) +
        ↪  '.txt')
    Hz = helper.bnToHash(IDz + Sz)
    assert g2*Hz*dz == g2hz1 + g2hz2 + g2hz3
```

```
def sendQuery(IDp):
    w3.eth.defaultAccount = w3.eth.accounts[4]

    IDp = 1
    IDHS0 = 1

    app.functions.requestData(IDp, IDHS0).transact()

def sendResponse(requestID):
    start = time.time()
    request = app.functions.requests(requestID).call()
    invoicesCount = app.functions.invoicesCount().call()

    [g1a1, g1a2, g1a3, g1b1, g1b2, g1b3] = [helper.strToG1Point(x) for x in app
        ↪ .functions.initData().call()]

    [x1, x2, x3] = helper.getKeysFromTextFile('keys.txt')

    Dz1_1 = 0
    Dz2_1 = 0
    Dz2_2 = 0
    Dz3_2 = 0
    Dz3_3 = 0
    Dz1_3 = 0

    for i in range(1, invoicesCount + 1):
        [IDz, IDp, dz11, dz21, dz22, dz32, dz33, dz13] = app.functions.
            ↪ invoices1(i).call()
        if IDz == 0:
```

```
    continue

[dz11, dz21, dz22, dz32, dz33, dz13] = [helper.intToBigNum(x) for x in
    ↪ [dz11, dz21, dz22, dz32, dz33, dz13]]
Hx1z = helper.bnToHash(x1 + IDz)
Hx2z = helper.bnToHash(x2 + IDz)
Hx3z = helper.bnToHash(x3 + IDz)

Hg1a1hx1 = helper.pointToIntHash(g1a1 * Hx1z)
Hg1b1hx1 = helper.pointToIntHash(g1b1 * Hx1z)

dz1_1 = (dz11 * Hg1a1hx1.mod_inverse(o)) % o
Dz1_1 += dz1_1
dz2_1 = (dz21 * Hg1b1hx1.mod_inverse(o)) % o
Dz2_1 += dz2_1

Hg1a2hx2 = helper.pointToIntHash(g1a2 * Hx2z)
Hg1b2hx2 = helper.pointToIntHash(g1b2 * Hx2z)

dz2_2 = (dz22 * Hg1a2hx2.mod_inverse(o)) % o
Dz2_2 += dz2_2
dz3_2 = (dz32 * Hg1b2hx2.mod_inverse(o)) % o
Dz3_2 += dz3_2

Hg1a3hx3 = helper.pointToIntHash(g1a3 * Hx3z)
Hg1b3hx3 = helper.pointToIntHash(g1b3 * Hx3z)

dz3_3 = (dz33 * Hg1a3hx3.mod_inverse(o)) % o
Dz3_3 += dz3_3
```

```
    dz1_3 = (dz13 * Hg1b3hx3.mod_inverse(o)) % o
    Dz1_3 += dz1_3

    assert dz1_1 == dz1_3
    assert dz2_1 == dz2_2
    assert dz3_2 == dz3_3
    assert (dz1_1 + dz2_1 + dz3_2) % o < 100

# post the data now
# just post three responses separately
w3.eth.defaultAccount = w3.eth.accounts[1]
D1121 = hex(Dz1_1) + '||' + hex(Dz2_1)
eD1121 = helper.encryptData(insuranceKey.public_key.to_hex(), D1121)
heD1121 = eD1121.hex()
tx_hash = app.functions.respond(requestID, heD1121).transact()
w3.eth.waitForTransactionReceipt(tx_hash)

w3.eth.defaultAccount = w3.eth.accounts[2]
D2232 = hex(Dz2_2) + '||' + hex(Dz3_2)
eD2232 = helper.encryptData(insuranceKey.public_key.to_hex(), D2232)
heD2232 = eD2232.hex()
tx_hash = app.functions.respond(requestID, heD2232).transact()
w3.eth.waitForTransactionReceipt(tx_hash)

w3.eth.defaultAccount = w3.eth.accounts[3]
D3313 = hex(Dz3_3) + '||' + hex(Dz1_3)
eD3313 = helper.encryptData(insuranceKey.public_key.to_hex(), D3313)
heD3313 = eD3313.hex()
tx_hash = app.functions.respond(requestID, heD3313).transact()
```

```
    w3.eth.waitForTransactionReceipt(tx_hash)


def getResponse(requestID):
    start = time.time()
    [_, _, _, _, responseCount] = app.functions.requests(requestID).call()
    # we want at least 2 responses to be able to retrieve the answer
    assert responseCount > 1

    parts = []
    for i in range(1, responseCount + 1):
        response = app.functions.viewResponse(requestID, i).call()
        decryptedString = helper.decryptData(insuranceKey.to_hex(), bytes.
            ↪ fromhex(response))
        [part1, part2] = decryptedString.split('||')
        helper.addResponseNoDuplicate(parts, helper.intToBigNum(int(part1, 16))
            ↪ )
        helper.addResponseNoDuplicate(parts, helper.intToBigNum(int(part2, 16))
            ↪ )

    total = sum(parts) % o

    return total
```

A.3 Helper Functions Code in Python

```
from eth_keys import keys
from eth_utils import decode_hex
from ecies import encrypt, decrypt
from bplib import bp
from web3 import Web3
from random import randint
from hashlib import sha256

def parameters():
    G = bp.BpGroup()
    g1, g2 = G.gen1(), G.gen2()
    e, o = G.pair, G.order()
    return (G, o, g1, g2, e)

def getKeyPair(privKeyString):
    priv_key_bytes = decode_hex(privKeyString)
    priv_key = keys.PrivateKey(priv_key_bytes)
    return priv_key

def getPublicKey(privKeyString):
    priv_key = getKeyPair(privKeyString)
    pub_key = priv_key.public_key
    return pub_key.to_hex()

def encryptData(pubKeyString, dataString):
    dataBytes = dataString.encode('utf-8')
    return encrypt(pubKeyString, dataBytes)
```

```
def decryptData(privKeyString, data):
    dataBytes = decrypt(privKeyString, data)
    return dataBytes.decode('utf-8')

def concatNumbers(num1, num2):
    return hex(num1) + '||' + hex(num2)

def splitNumber(concatNum):
    [a,b] = concatNum.split('||')
    return (a,b)

# def from_bytes (data, big_endian = False):
# if isinstance(data, str):
# data = bytearray(data)
# if big_endian:
# data = reversed(data)
# num = 0
# for offset, byte in enumerate(data):
# num += byte << (offset * 8)
# return num

# def pointToNumber(curvePoint):
# pointBytes = curvePoint.export()
# return from_bytes(pointBytes)

def pointToStr(curvePoint):
    pointBytes = curvePoint.export()
    pointStr = pointBytes.hex()
```

```
    return pointStr


def strToG1Point(pointStr):
    pointBytes = bytes.fromhex(pointStr)
    point = bp.G1Elem.from_bytes(pointBytes, bp.BpGroup())
    return point


def strToG2Point(pointStr):
    pointBytes = bytes.fromhex(pointStr)
    point = bp.G2Elem.from_bytes(pointBytes, bp.BpGroup())
    return point


def generateInvoiceValue():
    return bp.Bn.from_decimal(str(randint(50, 100)))


def bnToHash(num):
    intBytes = Web3.toBytes(num.int())
    hashedWord = sha256(intBytes).hexdigest()
    bnInt = bp.Bn.from_hex(hashedWord) % bp.BpGroup().order()
    return bnInt


def pointToIntHash(curvePoint):
    pointBytes = curvePoint.export()
    hashedWord = sha256(pointBytes).hexdigest()
    bnInt = bp.Bn.from_hex(hashedWord) % bp.BpGroup().order()
    return bnInt


def intToBigNum(i):
    return bp.Bn.from_decimal(str(i))
```

```
def getKeysFromTextFile(filename):
    with open(filename) as f:
        keyStrings = f.readlines()
    keyStrings = [x.strip() for x in keyStrings]
    keyInts = [bp.Bn.from_decimal(str(int(x, 16))) for x in keyStrings]
    return keyInts


def addResponseNoDuplicate(list, res):
    for i in range(0, len(list)):
        if list[i] == res:
            return

    list.append(res)
```

Vita

Candidate's full name: Son Khanh Luong

Bachelor of Information System, Hanoi University of Science and Technology, Hanoi, Vietnam, 2013.

Posters:
S. Luong and R. Lu, Blockchain-Based Privacy-Preseving Medical Insurance Storage System, Research Exposition 2019, Fredericton, Canada, April 12, 2019.

Awards:
Sasi Mohan Pal Memorial Prize by University of New Brunswick, 2018.