DOKUZ EYLÜL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

PRIVACY IN BLOCKCHAIN BASED INTERNET OF THINGS NETWORK

by

Şafak ÖKSÜZER

June, 2022

İZMİR

PRIVACY IN BLOCKCHAIN BASED INTERNET OF THINGS NETWORKS

A Thesis Submitted to the Graduate School of Natural and Applied Sciences of Dokuz Eylül University In Partial Fulfillment of the Requirements for the Master of Science in Computer Engineering

by

Şafak ÖKSÜZER

June, 2022

İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"PRIVACY IN BLOCKCHAIN BASED INTERNET OF THINGS NETWORKS"** completed by **ŞAFAK ÖKSÜZER** under supervision of **ASSOC. PROF. DR. GÖKHAN DALKILIÇ** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

……..
Assoc. Prof. Dr. Gökhan DALKILIÇ
Supervisor

……..	……..
Prof. Dr. Yalçın ÇEBİ	Asst. Prof. Dr. Ömer AYDIN
(Jury Member)	(Jury Member)

Prof. Dr. Okan FISTIKOĞLU
Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to my supervisor, Assoc. Prof. Dr. Gökhan DALKILIÇ, for his support and guidance at all times.

I would like to thank Dr. Enis KARAARSLAN, who constantly encouraged me and provided support for my thesis.

Finally, I like to express my sincere gratitude to my family, Gülçin ÖKSÜZER and Bülent ÖKSÜZER, who have always motivated and supported me during my thesis writing.

Şafak ÖKSÜZER

PRIVACY IN BLOCKCHAIN BASED INTERNET OF THINGS NETWORKS

ABSTRACT

The Internet of things devices have become a part of our daily lives. The data generated by the sensors on these devices is considered personal data, especially health data. For this reason, the data generated on IoT devices must be shared securely. In this thesis, a distributed and autonomous solution is presented to share the data generated by IoT devices with companies or customers in a limited period of time. For this purpose, the blockchain solution Quorum is used, which provides blockchain infrastructure for enterprises. Considering the low processing and storage capacities of IoT devices, an efficient and powerful key sharing system has been developed. All transactions in this system are recorded with a smart contract. There is such a temporary privacy sharing method with blockchain infrastructure for enterprises.

Thanks to the tamper-proof, distributed, and programmable infrastructure offered by blockchain technology, all sharing processes are securely stored in the nodes of the blockchain network. As the performance tests show, this sharing can be realized even with low-capacity IoT devices. These performance tests were conducted on a limited capacity device and the results were analyzed. This thesis also discusses potential problems of blockchain and IoT devices and proposes possible solutions.

Keywords: Quorum, blockchain, immutability, internet of things, privacy, cryptography

BLOK ZİNCİRİ TABANLI NESNELERİN İNTERNETİ AĞLARINDA MAHREMİYET

ÖZ

Internet of things cihazları günlük hayatımızın bir parçası haline gelmiştir. Üzerlerinde bulunan sensörler ile üretilen veriler özellikle sağlık verileri gibi kişisel veri olarak değerlendirilmektedir. Bu sebeple, IoT cihazları üzerinde üretilen verilerin güvenli bir şekilde paylaşılması gerekmektedir. Bu çalışmada, kurumsal bir blockchain altyapısı sağlayan Quorum blockchain çözümü kullanılarak IoT cihazlar üzerinde üretilen verilerin kısıtlı bir zaman diliminde şirketler veya istemciler ile paylaşılması için bir dağıtık ve otonom bir çözüm sunulmuştur. IoT cihazların düşük işlem ve hafıza kapasiteleri göz önünde bulunularak verimli ve performansı yüksek bir anahtar paylaşım sistemi geliştirilmiştir. Bu sistem içerisindeki tüm işlemler akıllı sözleşme ile kayıt altına alınmıştır. Kurumsal blockchain altyapısı kullanılarak böyle bir geçici mahremiyet paylaşımı yöntemi bulunmaktadır.

Blockchain teknolojisinin sağladığı tamper-proof, dağıtık ve programlanabilir altyapı sayesinde tüm paylaşım süreçleri blockchain ağı üzerinde bulunan düğümlerde güvenli bir şekilde saklanmıştır. Yapılan performans testlerinin de gösterdiği gibi, düşük kapasiteli IoT cihazlar ile bu paylaşım gerçekleşebilmektedir. Bu performans testleri örnek uygulama yapılan düşük kapasiteli kısıtlı cihaz üzerinde gerçekleştirilmiş ve elde edilen sonuçlar analiz edilmiştir. Bu çalışmada, ek olarak blockchain ve IoT cihazlarının potansiyel sorunları da tartışılmış ve olası çözüm önerileri yapılmıştır.

Anahtar kelimeler: Quorum, Blok Zinciri, değişmezlik, nesnelerin interneti, mahremiyet, kriptoloji

CONTENTS

Page

M.Sc. THESIS EXAMINATION RESULT FORM ii

ACKNOWLEDGEMENTS iii

ABSTRACT iv

ÖZ v

LIST OF FIGURES ix

LIST OF TABLES x

CHAPTER ONE -- INTRODUCTION 1

1.1 Privacy 2

1.1.1 General Data Protection Regulation 2

1.1.2 Personal Data Protection Authority 3

1.2 Internet of Things 3

1.2.1 Security and Privacy Challenges 4

1.3 Blockchain 4

1.3.1 Public Blockchain 6

1.3.2 Private Blockchain 6

1.3.3 Permissioned Blockchain 6

1.3.4 Permissionless Blockchain 7

1.3.5 Enterprise Blockchain 8

1.3.5.1 Hyperledger Fabric 9

1.3.5.2 R3 Corda 9

1.3.5.3 Avalanche 10

1.3.5.4 Quorum 11

1.4 Motivation 11

1.5 Contribution 12

1.6 Outline of the Thesis 13

CHAPTER TWO -- RELATED WORKS 14

2.1 Literature Review 14

CHAPTER THREE -- PRIVACY SHARING SYSTEM FOR INTERNET OF THINGS 18

3.1 Quorum 18
3.1.1 Quorum Network 19
3.1.2 Consensus Algorithms 21
3.1.2.1 Raft 22
3.1.2.2 IBFT 24
3.1.2.3 Clique 26
3.1.3 Ethereum Virtual Machine 28
3.1.3.1 Smart Contracts 29
3.1.3.2 Solidity 30
3.1.4 Transactions 31
3.1.4.1 Public Transactions 33
3.1.4.2 Private Transactions 34
3.1.5 Network 34
3.1.5.1 Quorum Node 35
3.1.5.2 Tessera Node 36
3.2 Keccak Hashing Algorithm 38
3.3 Recursive Length Prefix 39
3.4 Elliptic-Curve Cryptography 40
3.4.1 Generating Key Pairs 41
3.4.2 Elliptic Curve Digital Signature Algorithm 42
3.5 Public Key Recovery 44
3.6 Elliptic Curve Diffie-Hellman 46

CHAPTER FOUR -- IMPLEMENTATION 47

4.1 Experimental Setup 48
4.1.1 Quorum Network Specification 48
4.1.2 Smart Contract Specification 49
4.1.3 ESP32 Specification 52
4.2 Proposed Blockchain-Based Privacy Sharing System 52

CHAPTER FIVE -- ANALYSIS OF THE PROPOSED SYSTEM 61

5.1 Performance Analysis 61
5.2 Potential Security Attacks Analysis 65
5.2.1 Double-Spending 65
5.2.2 Consensus Algorithm Vulnerabilities 65
5.2.3 Node Capturing 66
5.2.4 Device Cloning 67
5.2.5 Privacy Keys Security 67
5.2.6 Man-in-the-Middle Attack 68

CHAPTER SIX -- CONCLUSION AND FUTURE WORK 69

REFERENCES 71

APPENDICES 79

Appendix 1: ESP32 Code 79
Appendix 2: Genesis File for Quorum Blockchain with Raft Consensus 85
Appendix 3: Smart Contract 86
Appendix 4: Register Company Transaction 89
Appendix 5: Register IoT Device Transaction 90
Appendix 6: Privacy Sharing Process ESP32-WROOM Logs 91

LIST OF FIGURES

Page

Figure 3.1 Sample genesis file 19
Figure 3.2 Example permissioned node 20
Figure 3.3 Example geth command to run a Quorum node 20
Figure 3.4 Example static-nodes.json file 21
Figure 3.5 Roles of Raft consensus algorithms 22
Figure 3.6 Raft election flow 23
Figure 3.7 Raft consensus algorithm static-nodes.json file example 23
Figure 3.8 The states of the IBFT algorithm 25
Figure 3.9 New block adding process in Clique consensus algorithm 27
Figure 3.10 Deploying a new contract flow 29
Figure 3.11 Example input data of an Ethereum transaction 31
Figure 3.12 High-level architecture of Quorum blockchain 35
Figure 3.13 Curve of secp256k1 standard 41
Figure 3.14 Sending raw transaction flow 44
Figure 3.15 ECDH algorithm flow 46
Figure 4.1 Main flow of proposed system ..... **Error! Bookmark not defined.**
Figure 4.2 Authorization flow via the smart contract 56
Figure 4.3 Privacy sharing flow 57
Figure 5.1 Keccak hashing algorithm performance of ESP32-WROOM 61
Figure 5.2 ECDSA performance of ESP32-WROOM 62
Figure 5.3 Secp256k1 algorithm performance of ESP32-WROOM 62
Figure 5.4 Secp256k1 algorithm performance of ESP32-WROOM & WiFi ON 63
Figure 5.5 Method performance of ESP32-WROOM 64
Figure 5.6 Sending transaction performance of ESP32-WROOM 64

LIST OF TABLES

Page

Table 4.1 Quorum nodes information 49
Table 4.2 Roles and authorization of the smart contract..... 51
Table 4.3 Example transaction for privacy sharing request..... 58
Table 4.4 Call example for checking existence of privacy sharing request..... 59
Table 4.5 Creating shared key..... 60
Table 4.6 Privacy sharing confirmation transaction 60

CHAPTER ONE
INTRODUCTION

Our world is undergoing a digital transformation. In this process, many devices in our daily lives are now digital. Internet of Things (IoT) technologies include billions of connected devices, and that number is growing by millions every year. Everything from smart watches we use in our daily lives to smart household products can be referred to as the Internet of Things. The IoT is basically a network structure that consists of electronic devices that can be connected to each other. These devices with low computing resources, which have small processors and sensors, can collect data through their sensors. They are usually lightweight devices that are less powerful than our everyday personal computers (Li et al., 2015).

Since these devices cannot store as much data due to their light weight, they need to send the data generated by sensors or other units on the devices to a trusted storage unit through a network connection. The network connection is important; it must be established using a secure protocol. There are several legal regulations that restrict the sharing or use of data as it can be considered personal data, such as Personal Data Protection Authority in Turkey or the General Data Protection Regulation (GDPR) in the European Union.

Classically, server-client architectures are used for IoT devices that communicate with the server to share or store their data. This method has its own security breach issues and contributes to increasing the possibility and severity of security problems. Therefore, a new way of integrating blockchain technology is on the rise to transform these centralized systems into decentralized systems without any authority and make them autonomous. Blockchain technology enables IoT devices to connect to each other through a peer-to-peer network and efficiently avoid single-point-of-failure and bottleneck problems. In addition, blockchain technology can reduce the cost of the current approach while handling the exponentially growing number of IoT devices (Kshetri, 2017).

Blockchain and IoT devices can be used together to create a better system for privacy. Each of them has some advantages that compensate for its weaknesses.

1.1 Privacy

The protection of personal data is directly related to privacy. Therefore, the privacy security of users using IoT devices must be ensured. Privacy protection includes keeping personal data secure and deciding what to do with it or how to process it. The right to privacy can be considered a human right or a personal right or property.

These two main laws are important to ensure privacy in the EU and Turkey. Personal Data Protection Authority (PDPA) and General Data Protection Regulation (GDPR).

1.1.1 General Data Protection Regulation

The General Data Protection Regulation (GDPR) was accepted and published in 2016 and has since continued to improve in line with new technological developments. It has been applied since May 24, 2018 and applies to anyone with a connection to the territory or market of the European Union. It may result in a fine of up to 4% of annual global turnover in the case of one company or up to €100 million in all cases. All Data Protection Authorities (DPAs) must enforce this and they must be able to take all precautionary measures to prevent the breach and leakage of personal information without control (Albrecht, 2016).

The GDPR contains six general data protection principles: Fairness and Lawfulness, Integrity, Purpose Limitation, Confidentiality, Storage Limitation and Data Minimization. The most important thing about the GDPR is data protection by design and default. This idea is supported by transparency, which means that all participants must have access to their information or the system should allow participants to control and monitor their data. In addition, all organizations that demonstrably work with personal data must be accountable. To be accountable, all

organizations must design their system to have a logging mechanism built into their systems. Thus, if there are issues with the protection of personal data, they must be prepared to show and explain their logging data (Goddard, 2017).

1.1.2 Personal Data Protection Authority

In Turkey, Personal Data Protection Authority (PDPA) operates similarly to the GDPR in the EU, but applies mainly to companies and businesses that provide a service in Turkey. It was adopted and announced on March 24, 2016. The general principles of the PDPA are in Article 4 of the Personal Data Protection Law numbered 6698 (Yücedağ, 2019).

The protection of personal data in digital media is of great importance, especially for PDPA. In particular, the security of IoT devices that generate data that can be considered personal data and that can transmit data to another location in some way also falls within the scope of PDPA. Specifically for PDPA, any data that can be used to draw conclusions about the user is considered personal data. The transfer and analysis of personal data is carried out through contracts to which users have agreed.

1.2 Internet of Things

Internet of Things (IoT) devices can be defined as a connectable lightweight device network, i.e., millions of heterogeneous devices containing sensors or actuators form a different kind of network. The network connectivity of these types of devices helps them themselves to collect and share data with various participants such as enterprises or governments. These devices have three main characteristic features that need to be categorized. The first is the limited computing capacity, which means that the computing power is very low compared to a normal personal computer. The second is limited storage capacity. The storage capacity of these devices is limited to store all the data they generate. Therefore, they need to be connected to a network to share the data. Connectivity is an essential requirement to be an IoT device. The last point is the limited processing capacity. Because of this characteristic, IoT devices cannot analyze the data they collect using sensors or other

things. A large storage server like a cloud is urgently needed to keep the IoT network alive.

Technology is developing rapidly and hardware solutions are becoming more available and sustainable. These developments mean that we can produce a wide range of devices. Smart watches, for example, which can be called IoT devices, offer so many different functions such as controlling heartbeat or blood pressure using minimal sensors. These types of devices have started to use our lives in a good way.

1.2.1 Security and Privacy Challenges

The security of IoT systems is important because, as mentioned earlier, these devices have become part of everyday life. Therefore, minimum security requirements are defined. These requirements can be listed as follows: confidentiality, integrity, availability, and authentication (Grammatikis et al., 2019).

In addition to the security challenges of IoT systems, there are various problems in the real world, such as the theft or copying of devices. Therefore, physical security in these systems is a serious matter. Every device in an IoT network must be physically protected. Human factors must be taken into account because people can damage or hack the devices for their own purposes. On the other hand, environmental factors must also be considered. In harsh climatic conditions, some devices may break or be inaccessible for too long. Every parameter is important in IoT systems, because these systems are like a bridge from the virtual to the real world, and this bridge is quite fragile.

1.3 Blockchain

Blockchain technology is not a completely new technology. Satoshi Nakamoto announced a new economic model when he published a paper called Bitcoin in 2009. This paper included several old-fashioned technologies such as peer-to-peer (P2P), asymmetric cryptology, and hashing. Although it included these technologies, the proposed system was a decentralized economic model. The new system was called

Blockchain and used the old technologies to create a new method for transferring assets or money.

Satoshi Nakamoto's Blockchain system includes the concept of smart contracts. Nick Szabo mentioned smart contracts back in 1997, and basically, smart contracts are just a piece of code that cannot be changed or updated by anyone once implemented. After an agreement between two or more sides, a smart contract can be deployed and each participant in the agreement must abide by the deployed contract. Thus, Bitcoin also has a smart contract to transfer currencies between users (Szabo, 1997).

Blockchain systems are decentralized systems with high cryptological security. All data generated by smart contracts is stored in a chain. The chain contains blocks as data. Each block is linked with its hash to the previous block. The link between blocks allows monitoring of changes or manipulations of data in the system. When old data is changed in an old block, the hash value is changed and the change can be easily detected. In addition, the data chain can be stored on different computers, which makes the blockchain a distributed system. Distributed systems have many advantages such as high availability.

The place where the data is stored in the blockchain is called the ledger. All nodes in the blockchain network use a common ledger because the data must be consistent. The ledger stores transactions that take place within the blockchain network. In some blockchain structures, nodes may contain more than one ledger, such as in the Quorum blockchain.

The main problem with the blockchain is deciding who will add the new block to the chain. This problem is a common problem of distributed and decentralized systems. The problem is called consensus problem. There are several consensus algorithms to solve this problem. These consensus algorithms are specialized for the different blockchain types. Essentially, there are two blockchain types based on the visibility of the data: Public and Private. In addition, depending on the process of

joining a blockchain network, there are two main blockchain types: permissioned and permissionless. These two categories can be combined as public-permissionless blockchain (Wüst & Gervais, 2018).

1.3.1 Public Blockchain

Public blockchain networks have no control mechanism to view the data on the chain. Anyone can see and monitor the data in the chain. Bitcoin and Ethereum are well-known examples of this type of blockchain. The chain can be downloaded and copied by anyone without permission. In the blockchain, users can keep the transaction confidential with their public keys. There is no connection between the public keys and the users unless they want to leak their information.

1.3.2 Private Blockchain

Private blockchain systems are the opposite of public blockchain systems. All data stored in the blockchain cannot be monitored or analyzed by unauthorized participants. Each participant has some privileges for monitoring. Each node that joins the blockchain network as a participant is authenticated and can only exist in the network with certain privileges. The privileges are set by the administrator of the network. In private blockchain networks, unlike public blockchain networks, the system is not fully decentralized. Authority is needed to control what participants can do. This makes the decentralized system a type of centralized system in more ways than one (Yang et al., 2020).

1.3.3 Permissioned Blockchain

In permission-requiring blockchain networks, joining a blockchain network requires the issuance of a permission. There must be an authority on the network that can allow anyone to become part of the network. In this type of network, decentralization and anonymity cannot be guaranteed. The authority in the network must identify the nodes as participants. During this identification process, a lot of information can be verified by the authority. Unlike permissionless blockchain like

Bitcoin, the participants in the network can be easily monitored and tracked. The stored data is also secured by the administrator. The system is still tamper-proof, but permission control allows control over the blockchain (Nadir, 2019).

For permissioned blockchain networks, consensus algorithms that require less energy are preferred. Because they involve trusted participants, unlike public and permissionless networks, the consensus algorithms used aim for high performance rather than building and ensuring trust (Wüst & Gervais, 2018).

1.3.4 Permissionless Blockchain

Permissionless blockchain systems can be considered as open networks that everyone can join. In these systems, there is no control mechanism to identify the participants. Bitcoin and Ethereum are well-known examples of permissionless blockchain networks. Everyone can be a node of Bitcoin or Ethereum blockchain network by downloading the software. Then, being a part of the consensus algorithm happens without asking permission.

Direct connection to a network has some advantages and disadvantages. For example, it requires the use of complex and energy-consuming consensus algorithms. This need is a disadvantage with this type of system. Also, there should be an award mechanism to keep all participants in the system and continue the process. Reward mechanisms can sometimes be cryptocurrencies like PoW in Bitcoin. In PoW, whoever finds the right answer receives Bitcoin as a reward. The advantage is that it is very easy to participate in these networks. So, this feature makes these systems more accessible to everyone. Ethereum has a large community because it works without permission. In addition to this advantage, decentralization can take place in permissionless networks. In permissionless networks, every participant has almost equal rights, so there is no authority. Without authority, these systems become open to %51 attacks in PoW. If someone can take over %51 of the nodes in permissionless networks, the system becomes vulnerable to attacks (Fullmer & Morse, 2018).

1.3.5 Enterprise Blockchain

Enterprise blockchain systems are essential for real-world business applications. High performance and low power are two essential components of enterprise ecosystems, and permissionless networks do not meet these requirements because they have consensus algorithms such as PoW and PoS. Therefore, consensus algorithms with high throughput and low energy consumption are required to be used in real business applications. Considering all these requirements, using permissioned blockchain systems with specially designed consensus algorithms is the right system for this purpose (Hamida et al., 2017).

Data in enterprise solutions is critical. In particular, enterprise data can be commercially valuable and have outstanding significance for the future. Therefore, keeping the data under lock and key and obtaining permission to monitor and view the data in enterprise systems is of great importance to most companies. Therefore, using public blockchain systems is not feasible for enterprise solutions (Hamida et al., 2017).

Some blockchain systems aim to become enterprise solutions by considering aspects such as data secrecy and high performance. In order to achieve high throughput, some companies develop specially designed algorithms and special protocols for communication between nodes in the network. To ensure data privacy, the enterprise blockchain has data restriction protocols that use various cryptological methods.

There are several enterprise blockchain systems, such as Hyperledger Fabric, Corda, Avalanche, and Quorum by Consensys. They have different approaches to enterprise needs by applying blockchain fundamentals. High performance with low power consumption is always their goal (Mazzoni et al., 2022).

1.3.5.1 Hyperledger Fabric

Hyperledger Fabric, a sub-project of Hyperledger, is a permissioned and private blockchain system. Initially, IBM and Digital Asses developed Hyperledger Fabric to meet the needs of an enterprise solution. Less power requirements and data sharing with permission are the goals of this project. It also requires time-consuming consensus algorithms for real-time applications (Androulaki et al., 2018).

Hyperledger Fabric, a subproject of Hyperledger, is a permissioned and private blockchain system. Originally, Hyperledger Fabric was developed by IBM and Digital Asses to meet the needs of an enterprise solution. Less power requirements and data sharing with permission are the goals of this project. It also requires time-consuming consensus algorithms for real-time applications (Androulaki et al., 2018).

The smart contract in Hyperledger Fabric is called 'chaincode' and can be developed in familiar programming languages such as GoLang and JavaScript. This approach has some advantages. For example, developers do not need to learn any other programming language to develop chaincode in Hyperledger Fabric. On the other hand, using a non-specific programming language can cause problems. Potential risks include not preventing randomness and using operating system methods (Yamashita et al., 2019). Hyperledger Fabric is also a modular system that includes different components for various purposes (Nguyen et al., 2021).

Hyperledger Fabric has a modular architecture for implementing consensus algorithms. A Hyperledger network can establish a crash fault tolerant (CFT) and a byzantine fault tolerant (BFT). In addition to this, Hyperledger Fabric also supports the Raft consensus algorithm since version 2.x (Cachin et al., 2016).

1.3.5.2 R3 Corda

R3 was established in 2015 as a consortium. This consortium includes hundreds of financial companies that contribute to blockchain and distributed ledger technologies. Then they decided to develop a new blockchain framework to make the

financial ecosystem decentralized and blockchain-based. To this end, the R3 consortium created Corda (Clincy & Shahriar, 2019).

Corda is an open-source blockchain framework that supports interoperable blockchain networks with strict privacy policies. Moreover, smart contracts in Corda can be like legal contracts, so they are not just codes (Mohanty, 2019).

Privacy is a primary concern in Corda, and the channeling system allows all data to be kept secure and private between participants in a Corda network. The channeling system is called "Vault." In addition to this feature, smart contracts can be developed using popular high-level languages such as Java and Kotlin. Corda Distributed Applications (CorDapp) require only one validation from each associated transaction. This type of validation provides improved performance (Mohanty, 2019).

1.3.5.3 Avalanche

Avalanche is an Ethereum-based blockchain system that is an open-source project for developing and deploying decentralized applications (Dapp) and providing enterprise solutions. The main characteristic features of Avalanche are being interoperable and providing high scalable ecosystem (Tanana, 2019).

Avalanche supports the newly created consensus algorithm with the same name as the system. The Avalanche consensus algorithm is the main difference from the other decentralized networks. The Avalanche consensus algorithm works by recursive voting of subsets of participants inthe network. It is a type of voting-based consensus algorithm. The most important feature of this algorithm is that it provides high performance with its probabilistic security guarantee. This algorithm, which is a sub-member of the Snow protocol family, provides an exponentially increasing transaction confirmation speed relative to the number of nodes in the network by responding to probabilities. It provides a scalable structure as it does not lose performance by increasing the number of nodes (Rocket et al., 2019).

1.3.5.4 Quorum

Quorum is a blockchain framework based on Ethereum. It was started as an open-source project by JP Morgan and then further developed by Consensys. Unlike Avalanche, Quorum offers a fully private and permissioned blockchain structure. Quorum was developed for enterprise solutions and includes many different features (Baliga et al., 2018).

The main feature of these functions are the systems that prioritize privacy. It includes features that aim to protect privacy within the permissioned blockchain network. Another feature that aims to protect privacy is the special channel-like structure within Hyperledger Fabric. The name of this structure is 'Private Transaction'. By using private transactions, privacy can be ensured between nodes in a Quorum network (Mazzoni et al., 2022).

Quorum, which provides a fully private and permissioned blockchain network function, offers a development environment that does not require cryptocurrency due to its structure. The main reason for this is the consensus algorithms such as Raft and IBFT. Since these algorithms are custom algorithms, they also meet the high-performance requirements for enterprise solutions. However, unlike Avalanche's algorithm, there may be performance degradation as the number of nodes in the network increases (Baliga et al., 2018).

1.4 Motivation

Whether it's smart watches, smart kitchen appliances or smart cars, IoT devices have become an essential part of our lives today. These devices generate big data about our daily lives with their sensors. This generated data is the users' confidential data. The reason why this data is private data is because companies can draw conclusions about users through analysis. What can be done as a result of the analysis is restricted by laws such as PDPA or GDPR. However, despite the laws, the necessary technical infrastructure has not yet been created to track this process in a transparent way.

With a blockchain-based system proposal, it is possible to track the data produced on IoT devices. Blockchain can eliminate authority by offering an autonomous and decentralized structure. Thus, it offers a supervisory mechanism independent of companies or institutions. The main motivation of this thesis is to monitor the sharing of data produced on users' devices with a decentralized structure. In summary, this thesis aims to provide the technical infrastructure to ensure that the laws are fully implemented. For this purpose, the system will be autonomously sustainable without cryptocurrency by using Quorum.

1.5 Contribution

The contribution of this study is to ensure that IoT devices can communicate directly over a blockchain network. Although IoT devices have little hardware, they are connected to a blockchain network to send transactions. These processes connect to the Quorum network without an intermediate layer.

The consequence of the request to share the information generated on the device is stored in the blockchain. In this way, all transactions for sharing personal information are recorded in the blockchain. The recording processes are performed in a closed network.

Since the Quorum infrastructure is used, powerful consensus algorithms provide the performance required in real-world scenarios. Another advantage is that these consensus algorithms do not require cryptocurrencies. High performance and low energy consumption are provided together.

The tests were performed on a low-equipped ESP32-WROOM device. The ESP32 Web3 library was developed to connect ESP32-WROOM devices to the Quorum network. By using this library, ESP32 devices can send transactions to a smart contract on the Quorum blockchain network. During testing, the energy consumption of hashing algorithms such as SHA256 and Keccak was observed on the device. In

addition to these algorithms, the energy consumption of transactions sent directly through the device was also monitored.

1.6 Outline of the Thesis

The introduction addresses privacy issues related to the processing and sharing of data generated by IoT devices. In the second chapter, studies that address the concept of privacy in IoT technology are collected and the necessary explanations are provided.

In the third section, the technologies and concepts required for the proposed method of using blockchain technology for privacy sharing in IoT devices are explained in detail.

In the fourth section, the proposed system, the privacy sharing method, is explained in detail and step by step.

In the fifth chapter, the performance analysis of the proposed system is conducted. In addition to the performance analysis, solutions to potential problems of blockchain technology and IoT technology are also discussed.

In the sixth and final chapter, the results of this thesis are explained and what can be done in the future beyond this study is mentioned.

CHAPTER TWO
RELATED WORKS

There are studies conducted to protect privacy violations both legally and systematically. In these systems, the studies using blockchain and IoT devices together are directly related to our work. Our work is more concerned with tracking security breaches than preventing them. With this concern, we specifically looked at studies involving low-capacity IoT devices using private permissioned blockchain systems.

2.1 Literature Review

Mohanta et al. (2020) is in the process of recommending a new system to avoid security and privacy issues in IoT devices. In their proposal, token-based authentication in asynchronous communication between a blockchain-based distributed network and an IoT application provides privacy protection. In their experimental setup, a DHT22 sensor in a Raspberry Pi device was used to collect data in a room. The web3.js library is used to send the sensor data to the Ethereum network via transactions. This Ethereum network was not a real Ethereum network. They used Ganache, a local Ethereum network creation tool for testing purposes, to simulate the Ethereum network. "Decauth" authentication technique was the authentication method in their system to authenticate IoT devices in the network.

In their work, they addressed some security and privacy issues in IoT systems, such as secure unique identification, secure communication, privacy, and authorization. Some solutions were proposed for these problems. For example, they proposed using smart contracts as a self-executable program for privacy and authorization. In their view, smart contracts play a crucial role in privacy security. Thus, they suggested that authentication and authorization should be performed on a blockchain network to ensure security and trust. Also, only new devices or new users should be integrated into the blockchain to reduce the number of transactions. The

device used for testing can be considered a computer and not an IoT device. The Raspberry Pi has more computing power and memory than a normal IoT device.

Yuan et al. (2020) used the hash and CE -2 (HCE2) scheme to improve a blockchain-based open auditing and secure deduplication system. They proposed a fair arbitration system based on the HCE2 scheme. Their HCE2 scheme was designed to support auditing of data without the need for an external auditor. In addition, their proposed system was designed to automatically resolve data integrity issues using smart contracts.

They created the system in the cloud so that it can be accessed from anywhere. Their study aims to generate the same key by encrypting sensitive and important information using the HCE2 scheme. By using the HCE2 scheme, the storage and computation overhead of cloud service providers has been reduced. However, in the proposed scheme, a mining process is provided which increases the cost. The mining process is an important requirement for depositing a security deposit in public blockchain systems. Their system was based on the feature of HCE2 that the system can generate the same encryption key with the same data. So, if a user encrypts the sensitive data with HCE2 algorithm and then uploads it to a cloud. The system can use the same key to decrypt the uploaded data. If the system cannot decrypt the data with the same key, it means that the data has been modified.

Yu et al. (2018) mentions the use of Ethereum for authentication and access control of smart devices. The data in IoT devices can become more secure and available by using decentralized systems instead of traditional centralized systems. In their proposed solutions, users can set roles and privileges for the data using smart contracts. The transactions include some access management policies to store the information in a blockchain. Therefore, each policy can be requested for control and verification. In addition to access policies, private blockchain and consortium blockchain systems are proposed for privacy protection in IoT systems. The characteristics of blockchain, such as achieving anonymity and public keys, provide privacy protection for entities in the systems. Moreover, a zero-knowledge proof and

zero-knowledge argument were presented in the study to protect all information in a transaction from third parties.

Liu et al. (2021) aimed to provide a blockchain-based distributed access control system for the privacy of data in IoT devices. The method of their proposed system uses mixed linear and nonlinear spatiotemporal chaotic systems (MLNCML). Beside MLNCML, the least significant bit (LSB) to encrypt the data. After encryption, upload the encrypted data to a cloud system. The LSB method is an information hiding method in which the least significant bit of a digital entity, such as a colored image, is removed and replaced with another piece of information. The most important thing about the LSB method is that the overall quality of the altered digital entity is preserved. LSB and MLNCML were combined in the study to enable the exchange of information in a privacy-friendly manner over a blockchain network. Casbin Access Control Framework was used in the proposed system with Hyperledger Fabric for experimental purposes.

Zhang et al. (2021) proposed a data security sharing model based on privacy protection (DSS-PP) for the blockchain-based industrial Internet of Things. The proposed system requires a certificate authority. Data providers and data takers use a certificate authority to authenticate and register. After obtaining permission from a certificate authority, they enter the information into a blockchain system for sharing. Blockchain was used as a data storage entity. In addition to the blockchain, an off-chain database stores the original data by encrypting it with the key of the certification authority. This method of data storage created an efficient storage structure. Only the index of the original data was stored in the blockchain, which creates a lot of space in the main chain. In addition, the system provides high security because the original data is encrypted on the data outside the chain.

In the healthcare sector, this study (Srivastava, 2019) uses blockchain technology with IoT devices to protect patient privacy. Users can enter and store patients' health data encrypted in a remote monitoring tool. Then, a smart device as an IoT device summarizes the data by formatting it. Then, the system sends a smart contract in a

public blockchain. In this study, a new blockchain system called GHOSTDAG was developed, which includes a public blockchain and a private blockchain simultaneously. GHOSTDAG is a protocol that is a variant of the PHANTOM protocol. This protocol uses a directed acyclic graph (DAG) to build the blocks of the blockchain. The blocks are kept as clusters, so there can be many clusters. To find a block in these clusters, GHOSTDAG uses a greedy algorithm. The use of a greedy algorithm is the key difference with PHANTOM. After 3 clusters, PHANTOM may have a feasible problem, but using a greedy algorithm in the GHOSTDAH protocol made the systems more suitable for real-world scenarios. In the study, IoT devices were used to warn or inform patients.

Loukil et al. (2021) proposed a blockchain-based behavior detection system for privacy security in IoT device networks. The private Ethereum blockchain was used to avoid the use of cryptocurrencies. Three different smart contracts, such as Ownership Smart Contract, Privacy Permission Setting Smart Contract, and Behavior Control Smart Contract, control misbehavior in the system. The system involves many smart devices in a smart home or an enterprise such as a hospital. The Ownership Smart Contract records the ownership of the IoT devices in the system and has multiple methods to manage the devices. The Smart Contract Privacy Permission Setting stores the privileges and each smart device can query its privileges to communicate with other devices in the system. The smart contract behavior control is the core of the proposed system. The smart contract can be created by the data owner to define and store privacy settings. The privacy settings are used to check the permissions before devices are allowed to communicate with other devices, and the device outputs also depend on these privacy settings. When misbehavior of an IoT device is detected, the log of the detection is written to the blockchain via smart contracts. To detect misbehavior, the system checks the permissions by checking the privacy settings of the devices. If a problem occurs during the authorization check, a waiting period is set as a penalty to prevent a retry. Similarly, if repeated attempts are made, the penalty time is extended.

CHAPTER THREE
PRIVACY SHARING SYSTEM FOR INTERNET OF THINGS

Blockchain technology can be a great combination with IoT devices. To protect the privacy of personal data, which is one of the critical problems of today, a distributed system design can be created without an authority. To this end, it is possible to design an autonomous system that does not require a trusted third party to share personal data generated on IoT devices using blockchain technology. In this section, we detail how privacy is protected by using low-capacity IoT devices together with a private and permissioned blockchain structure as Quorum.

3.1 Quorum

Quorum is a private and permissioned blockchain system designed for enterprise solutions. The main requirement for enterprise solutions is high throughput performance and ease of management while maintaining security. Quorum meets these conditions by incorporating special consensus algorithms such as Raft and IBFT (Prusty, 2018).

Quorum was designed and developed by JP Morgan in 2018 through forking Ethereum. Then, in 2019 ConsenSys acquired Quorum from JP Morgan to add it into their other blockchain projects like Metamask (Mazzoni et al., 2022).

So basically, Quorum is an Ethereum fork that has some Ethereum features. The Ethereum blockchain was created to provide a blockchain playground that allows anyone to develop smart contracts without having to get permission. So, Quorum has some advantages of Ethereum, the use of Ethereum tools like Metamask is possible. It also has some special features, such as not requiring cryptocurrencies. However, one of the most important features is that there are some methods developed to ensure privacy in itself. For enterprise solutions, privacy security is important and crucial. Private transactions are used for this purpose. Quorum's smart contract architecture provides private data segmentation with private transactions. With

public transactions, Quorum can continue to use the features of Classic Ethereum (Polge et al., 2021).

3.1.1 Quorum Network

Quorum's architecture meets all the requirements placed on enterprise solutions. In particular, special consensus algorithms such as Raft are supported. Network management is also easy to handle. Adding nodes to networks or removing nodes from networks is quite simple. In addition, storing blocks with Merkle, as with Classic Ethereum, ensures effectiveness.

```
{
"alloc": {
"0xed9d02e382b34818e88b88a309c7fe71e69f419d":{
"balance": "1000000000000000000000000000"}},

"coinbase": "0x0000000000000000000000000000000000000000",
"config": {    "homesteadBlock": 0, "byzantiumBlock": 0, "constantinopleBlock": 0,
"chainId": 10, "eip150Block": 0,
"eip155Block": 0,
"eip150Hash":"0x0000000000000000000000000000000000000000000000000000000000000000"
"eip158Block": 0,
"maxCodeSizeConfig": [
{   "block": 0, "size": 35 } ],
"isQuorum": true },

"difficulty": "0x0",
"extraData": "0x0000000000000000000000000000000000000000000000000000000000000000",
“gasLimit": "0xE0000000",
"mixhash": "0x00000000000000000000000000000000000000647572616c65787365646c6578",
"nonce": "0x0",
"parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
“timestamp": "0x00"
}
```

Figure 3.1 Sample genesis file

When creating a quorum network, the genesis file is used. This file contains information such as some predetermined Ethereum wallets. The genesis file contains all the necessary information to create an Ethereum network. Since Quorum is an Ethereum fork, there are some settings in the Genesis file that are specific to the Quorum blockchain structure. Some of these settings are used to decide which consensus algorithm to set up and which block to start with. Also, how much data

each block can store is determined in this genesis file. An example genesis file with default settings is shown in Figure 3.1.

Within the genesis file, predefined smart contracts can also be added when creating the Quorum network.

As for the management of the network, there are mechanisms in the Quorum architecture to control the nodes in the network. These mechanisms are like any other blockchain structure with permissions. Each node in the network has its own remote key. These keys are stored in permissioned-nodes.json with port and IP address information. This file is up to date on each node, so each node recognizes other networks. The process of adding a new node to the network varies depending on the consensus algorithms.

[enode://6598638ac5b15ee386210156a43f565fa8c48592489d3e66ac774eac759db9eb52866898cf0c5e597a1595d9e60e1a19c84f77df489324e2f3a967207c047470@127.0.0.1:30300]

Figure 3.2 Example permissioned node

Permissioned nodes structure is *enode://REMOTE_KEY@IP_ADDRESS:PORT.* In Figure 3.2, an example permissioned node is shown.

```
geth
--wsapi "istanbul"
--rpcapi "istanbul"
--istanbul.requesttimeout 5000
 --allowedfutureblocktime 5
dumpconfig > config.toml
```

Figure 3.3 Example geth command to run a Quorum node

In addition to the genesis file, each node on the network can have its own settings. These settings are like the basic settings of the Go Ethereum (*geth*) structure. For example, the node's consensus algorithm port or WebSocket port can be used to

interact with the node, or how many nodes will be on a network. A node can be started by setting the basic settings with the *geth* command. An example *geth* command for running a Quorum node is shown in Figure 3.3.

Static nodes are considered trusted nodes. They are defined in the static-nodes.json file when the network is created. Example static-nodes.json is shown in

Figure 3.4. Static nodes are not counted against the maximum number of nodes defined when the network is created. Quorum periodically confirms the connections of static nodes and tries to stay connected all the time. Static nodes are crucial for consensus algorithms. After the network is created, new static nodes can be added, or old nodes can be removed from the network.

```
enode://cea71cb65a471037e01508cebcc178f176f9d5267bf29507ea1f6431eb6a5dc67d086dc8dc
54358a72299dab1161febc5d7af49d1609c69b42b5e54544145d4f@127.0.0.1:30303,
enode://ca05e940488614402705a6b6836288ea902169ecc67a89e1bd5ef94bc0d1933f20be16bc88
1ffb4be59f521afa8718fc26eec2b0e90f2cd0f44f99bc8103e60f@127.0.0.1:30304
```

Figure 3.4 Example static-nodes.json file

3.1.2 Consensus Algorithms

As of version v22.1.1, four different consensus algorithms are supported in Quorum. The four consensus algorithms are essentially the Proof of Authority consensus algorithm. The supported consensus algorithms are as follows: Istanbul Byzantine Fault Tolerance (IBFT), QBFT, Raft, and Clique. The consensus algorithms within the Quorum architecture have a pluggable structure. Once a Quorum network is created, the consensus algorithm can be replaced with another consensus algorithm under the right conditions. This process of changing the consensus algorithm is also called the migration process.

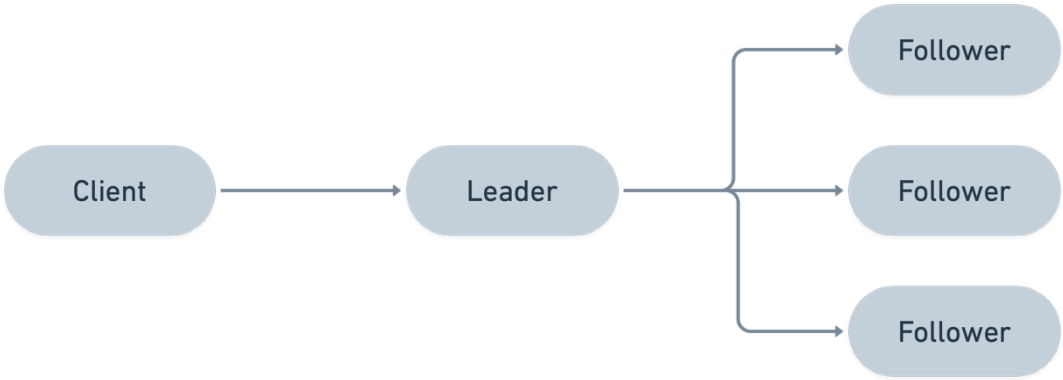


Figure 3.5 Roles of Raft consensus algorithms

3.1.2.1 Raft

The Raft protocol was developed in 2014 as an alternative to the Paxos algorithm, which is its predecessor. Basically, Raft is a voting-based consensus algorithm, meaning that there is a voting process (Ongaro & Ousterhout, 2014). In addition, Raft includes several roles for the participants in the consensus process. This relationship between roles is shown in Figure 3.5. These roles of nodes or, in other words, participants are (Huang et al., 2020):

Leader node: The leader of the consensus algorithm, chosen by the other nodes. Once selected or elected, the leader node can interact directly with the client and decide which blocks to add to the chain. When a new leader is selected, all other nodes synchronize with the leader node. The leader node is the authority in a term. The term is the period between two elections. The flow of the selection made to select the leader node is shown on Figure 3.6.

Follower node: Follower nodes follow the leader node to synchronize their copy of the data. Their main goal is to maintain data integrity with their copies. If the leader node is unavailable for some reason, one of the follower nodes can become the next leader node after winning an election.

Candidate node: If the leader node fails, there can be multiple candidate nodes. Nodes can ask other nodes for votes this is a voting request. When a node starts

sending vote requests to other nodes, that node is called a candidate node. If a candidate node can gather more votes than half of the nodes, it becomes the next leader node.

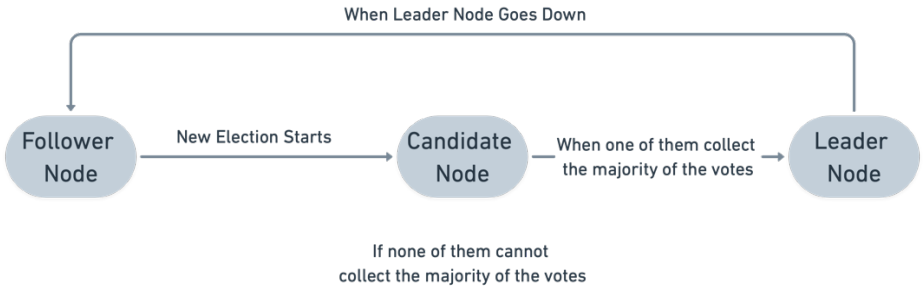


Figure 3.6 Raft election flow

Raft in Quorum is a special raft consensus algorithm. For a new transaction, it contains a new program segment every 50ms by default. Transitions of blocks in the Raft transport layer during HTTP state. As part of the Quorum, port 50400 is reserved for the Raft protocol. The maximum number of peers is 25, default.

Configures in Raft with views in *geth*:

raftport: Raft communication port
raftblocktime: New block adding frequency (ms)
maxpeers: Maximum peer count

Example static-nodes.json for Raft is shown in Figure 3.7.

```
[enode://6598638ac5b15ee386210156a43f565fa8c48592489d3e66ac774eac759db9eb52866898cf0c5e5
97a1595d9e60e1a19c84f77df489324e2f3a967207c047470@127.0.0.1:30400?raftport=50400]
```

Figure 3.7 Raft consensus algorithm static-nodes.json file example

A Quorum network can be migrated from the consensus algorithm to the QBFT algorithm. The Raft algorithm is not recommended for a production application due to its voting mechanism. Therefore, the QBFT algorithm is the better solution for production after development.

3.1.2.2 IBFT

IBFT is an extended variant of the Practical Byzantine Fault Tolerance (PBFT) algorithm. It is specifically designed for confidential and traceable blockchain structures. The main difference between IBFT and the Raft algorithm is that there are no follower nodes that completely follow the leader node. In the IBFT algorithm, it is assumed that there can be malicious nodes. Structurally, it has a validation process like proof of work. Byzantine fault tolerance is a type of tolerance mechanism. According to this mechanism, some nodes may be unreliable or malicious, so things must work for consensus to continue (Moniz, 2020).

The number of faulty nodes that can be tolerated under BFT is denoted by f. Thus, for the system to function properly, there must be 3f+1 nodes in the network. For a proper PBFT application, the BFT properties apply. According to this equation, about 1/3 of the nodes in a system can be tolerated (Moniz, 2020; Baliga et al., 2018).

The IBFT algorithm has the BFT property. Only the confirmation mechanism differs from the PBFT algorithm. The IBFT algorithm creates blocks at fixed intervals, regardless of pending transactions. For this reason, some blocks may not contain any transactions at all.

The approval mechanism of the IBFT algorithm is round-based. There is a pool of validation nodes called validators. The validators are responsible for validating new blocks. One of the validators becomes the proposer, which creates and organizes a block at a constant time and distributes the new block to other nodes. Then the validators check the new block and give their approval to add it to the chain. The most important point in this process is reaching a super majority, i.e. about %66 of the validators (Ongaro & Ousterhout, 2014).

There are 4 states in the IBFT algorithm:

Awaiting Proposal: Validators wait for the new block to be created and distributed by the proposer. When a new block can be created, the proposer proposes the block and transmits it in a pre-prepare message.

Preparing: Valid proposed block received, then validators are notified then the algorithm waits for their acceptance.

Ready: Once accepted by the validators, the proposed block is 'locked in' so that it cannot be changed or tampered with until it is inserted.

Round Change: When the process is finished for any reason, such as timeout, failure, or insertion, the algorithm waits for all validators to determine the next round number.

The transitions between the states of the IBFT algorithm are shown in Figure 3.8.

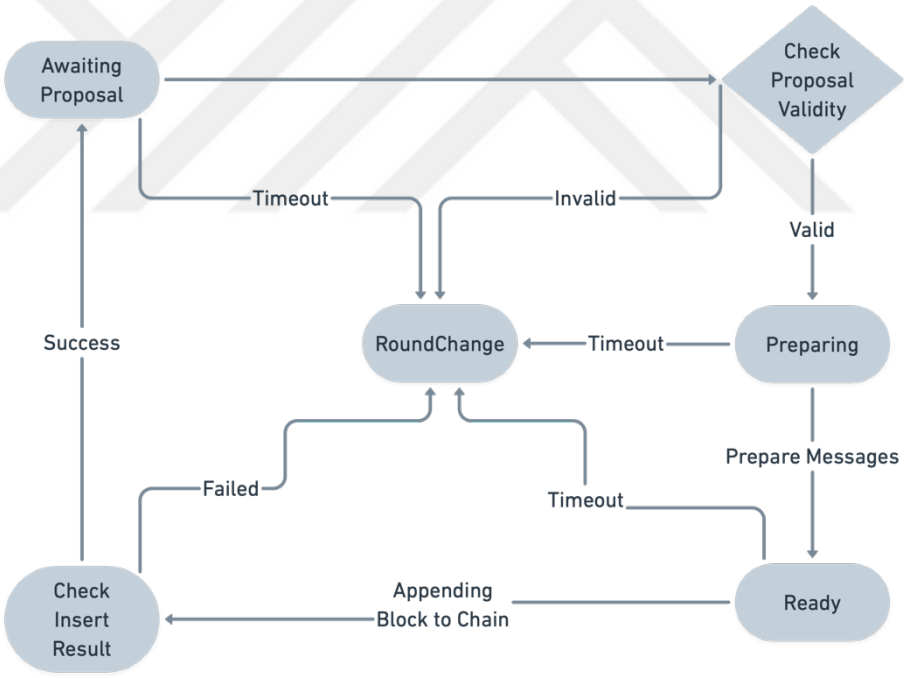


Figure 3.8 The states of the IBFT algorithm

The advantages of the IBFT algorithm are that it reduces the block validation time. Since it is possible to determine the block addition time, the number of transactions can be adjusted depending on the situation. The IBFT algorithm has a flexible structure. The set of Validators can be expanded by adding new trusted nodes after a network is created. Since a super majority is required to add a new

block, high data integrity is ensured. Although it is a leader node, it has high fault tolerance because the validation mechanism is executed with a group of trusted nodes.

In Quorum, the IBFT consensus algorithm has some special sections in the Genesis file.

epoch: Number of block that should pass before pending validator votes are reset
policy: Proposer selection policy, Round Robin or Sticky
ceilNby3Block: Block number, from where start to use the updated formula to calculate the faulty nodes count
extraData: RLP encoded string with a list of Validators

Configures in IBFT with views in *geth*:

istanbul.blockperiod: Block time
istanbul.requesttimeout: Timeout time for cancelation

3.1.2.3 Clique

Clique is a PoA consensus protocol. It was developed to make the Ethereum testnet more efficient. The first Ethereum testnet ran from July 2015 to about November 2016 and was called Morden. There was a lot of scrap and unused material in Morden. Also, in Morden, there were some consensus issues in the testnet between Geth and Parity. For this reason, Morden was discontinued to create an entirely new test network. Ropsten was developed by removing everything unnecessary from the network. The Ropsten test network started with basic PoW, but some malicious participants of the test network decided to abuse the basic PoW and inflate some prices in the system. In February 2017, Ropsten ceased operations due to the weakness of the PoW algorithm. The root of these attacks was based on the PoW algorithm, so starting a new test network from scratch was not a solution. Therefore, it was decided to develop a new consensus algorithm for Ethereum (Ahmad et al., 2021; Iyer & Dannen, 2018).

The main goal of the PoA protocol is to create a qualified development environment in an Ethereum test network. The new PoA protocol is based on the simple idea that new blocks can be added to the chain by trusted signers. The trusted signers also have known authorities. They are selected by the administrator of the network and can sign the new blocks (Zhang et al., 2019).

Clique was implemented in *geth* and therefore Quorum is a fork of *geth*. Quorum also has the Clique consensus algorithm. In a Clique-based Quorum network, there are multiple authorities. These authorities are responsible for approving the addition of a new block to the chain in a round-robin fashion. One authority produces the next block at a fixed interval and distributes it throughout the network, and the current authority releases it (Polge et al., 2021).

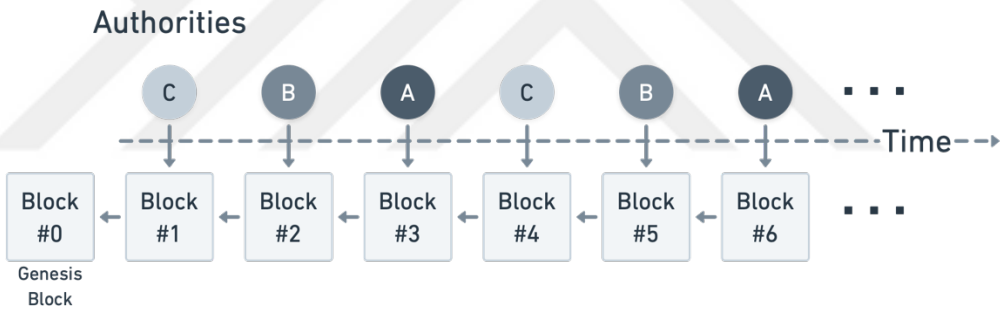


Figure 3.9 New block adding process in Clique consensus algorithm

Block period is calculated using the modulus operator (%), Block Number % Signer Count indicates the current authority signer. In the given example, there are three authorities in the order A, B, C. So, the signer of Block #2 is determined by 2 % 3, and after this calculation the signer of Block #2 is the second signer as B. This process is shown in Figure 3.9 (Clique, 2022).

Each block contains an *extraData* field that stores the authority signature in the header. This field has 32 bytes for vanity data, 65 bytes for the authority signature, and also contains the list of signers (Clique, 2022).

The use of the Clique consensus algorithm in enterprise solutions for private networks is not recommended due to its technical capacities.

3.1.3 Ethereum Virtual Machine

Ethereum is an open and distributed virtual computer which is the Ethereum Virtual Machine (EVM). Programs that can run on the EVM are called Ethereum Smart Contracts. At the same time, these programs are also called decentralized applications (Dapp). A program installed on the EVM, i.e., a smart contract, cannot be subsequently updated or modified. For this reason, applications are expected to be published to be reliable (Dannen, 2017; Mohanty, 2018).

All operations that can be performed on the EVM, such as issuing a smart contract or interacting with a smart contract, consume computing power. Every code that is executed on the EVM has a cost accounting. The unit of this cost accounting is called gas. The gas charge on the Ethereum mainnet is borne by the account that performs the transaction. However, with Quorum and similar institutional Ethereum fees, there is no gas fee. Although there is no gas fee, a gas amount limit can be implemented. The reason for this is that the transactions carried out by the EVM must not take too long (Mohanty, 2018).

Smart contracts can be developed using the Solidity programming language. Smart Contracts that are to be installed on the EVM must be compiled. The compiled smart contracts, also called bytecode, are stored on the EVM and each smart contract has an address. Through these addresses, accounts within Ethereum can interact with a smart contract. As a result of the interaction, a state change can occur. The states of the variables defined in the smart contract are stored on the EVM (Dannen, 2017).

The EVM exhibits deterministic behavior. Accordingly, a smart contract code executed in the EVM always leads to the same result. In a distributed and decentralized Ethereum network, the same Smart Contract with the same inputs is expected to produce the same results on all nodes. Otherwise, an inconsistency would cause the system to crash (Hirai, 2017).

3.1.3.1 Smart Contracts

Parts of the code that run on the EVM are called smart contracts. Smart contracts differ from regular programs in some respects. Not only do they have a non-updatable structure, but they also have no mechanism for not functioning on their own if they are not executed. A method within a smart contract can be called by another smart contract. In other words, Smart Contracts within the EVM can interact with each other. However, the data storage locations of each smart contract are different (Dannen, 2017).

There are some smart contract standards in Ethereum. These standards act as interfaces. The most well-known smart contract standard is the ERC-20 smart contract standard. For example, ERC-20 is a standard for implementing tokens that can be executed within the EVM. According to this standard, the method and variable names that a smart contract needs to be a token are specified. Thus, Ethereum token infrastructures such as USDT and EOS have an abstract structure and can be easily integrated anywhere (Garewal, 2020).

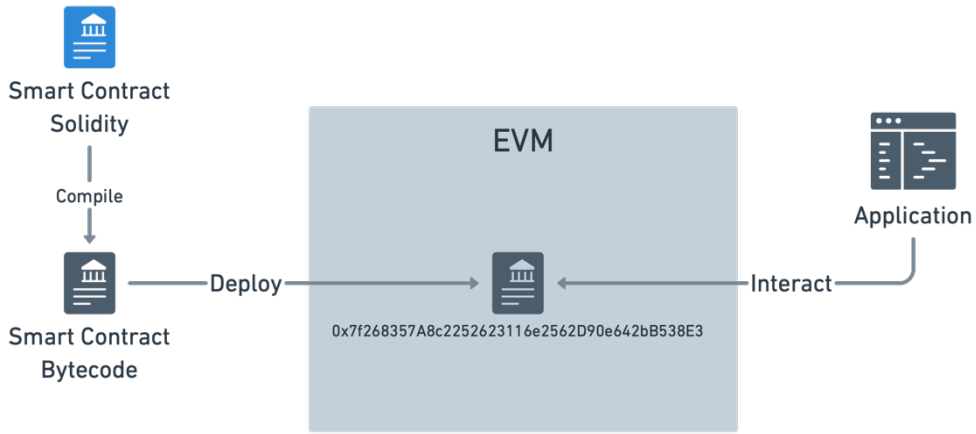


Figure 3.10 Deploying a new contract flow

As shown in Figure 3.10 after deploying a contract, a normal application can interact with the deployed contract throughout its determined address.

This interaction can occur in two ways. In the first way, a stage change takes place, i.e., a data update or addition can be made, such as changing a value in the smart contract. This type of interaction is called a transaction. In each transaction, there is information about which method is called with which parameters. There is also a gas calculation according to the operation of the called method. The second type of interaction is called Call. A data change in this interaction is just the process of reading data. Transactions are stored in the block, while call transactions are not stored in the block. Call operations are a type of data query operations (Tikhomirov, 2017).

3.1.3.2 Solidity

Solidity is a high-level programming language for developing Dapps with smart contracts in the Ethereum environment. It is an object-oriented and curly-bracket language, influenced by C++ and JavaScript (Schmeelk et al., 2021).

The main purpose of the Solidity programming language is to develop deterministic programs. Solidity is a domain-specific programming language designed exclusively for developing compilable smart contract code for EVM. It allows any smart contract executed on the EVM to produce the same results with the same inputs. Accordingly, randomness is inherently not allowed in a program written in Solidity, i.e., in a smart contract. It also does not allow the use of variables that depend on the device on which the code is executed, which could prevent determinism. In this respect, it differs from other platforms such as Hyperledger Fabric or Corda with the smart contract method. This is because other languages allow the use of languages that are already known and used for other purposes, such as Java or JavaScript, instead of a domain-specific high-level language like Solidity. However, the development of codes to run on the EVM is limited to Solidity (Mohanty, 2018).

```
0x00a469170000000000000000000000000000000000000000000000000000000000
00db6bd5000000000000000000000000000000000000000000000000000000000000
000000

Function: claimAll(uint256 blockNumber, bool toStrongPool)
MethodID: 0x00a46917
[0]:
0000000000000000000000000000000000000000000000000000000000db6bd5
[1]:
0000000000000000000000000000000000000000000000000000000000000000
```

Figure 3.11 Example input data of an Ethereum transaction

Solidity is a static-type language, but it also has features such as inheritance or interfaces because it is an object-oriented language. Structurally, it also allows the use of pre-built libraries. It is possible to develop very complex smart contracts with Solidity (Schmeelk et al., 2021).

After the code written with Solidity is compiled with Solidity Compiler, the bytecode for EVM is obtained. During this compilation process, the codes of the operations specified in EVM are determined, these codes are called opcodes. Each opcode is written in hexadecimal form. For example, '0x52' codes are used for MSTORE operation or '0x55' codes for SSTORE operation.

After the bytecode is deployed onto the EVM, the methodID associated with the method is used as a pointer for the method of a smart contract to be invoked with a transaction. Each methodID is unique as an eight-digit hexadecimal number. Methods that are invoked are stored in input data fields of transactions with parameters. An example input data is shown in Figure 3.11.

3.1.4 Transactions

Smart contracts published on the EVM can be structurally interacted with in two different ways. One is called Call and one is called Transaction. Specifically for Quorum, there are 2 different Transactions. Quorum includes EVM features as it is based on Ethereum, but also has features that are suitable for enterprise solutions. One of the most important features of Quorum is that it supports two different

transactions. These are public and private transactions. The fields of the transaction structure in Quorum are explained below (Baliga et al., 2018).

- from: The account address who sent the transaction

- to: The destination address of the transaction. It is empty or null when a new contract is deployed. Because there is no address for deploying the contract process, EVM can dedicate deploying transactions this way.

- value: It refers to the amount of sending money, but in Quorum it is not used.

- gas: The dedicated amount of gas to run the transaction. If some gas remains after running a transaction, the unused gas is refunded.

- gasPrice: The price of gas in a cryptocurrency but in Quorum the value of it is zero

- input: The data is for the purpose of the transaction. It contains bytecode while a new contract deploying. When executing a method in a deployed contract, it includes needed data of an initialization code.

- nonce: It is a number that is used to avoid double-spending problem. The nonce value is unique for each account in the network and is incrementally increased. If a transaction is sent with the same nonce value, Quorum will not accept that transaction. Instead of increasing the nonce value by one, transactions with high nonce values can also cause problems. It may wait until the transactions arrive before the nonce values of such transactions. In the Quorum network, pending transactions can sometimes be overridden by sending the same nonce value.

- v, r, s: Elliptic Curve Cryptology is used to sign a transaction, the output of these operations is stored in these fields. In addition, in Quorum v fields are used to indicate private transactions by setting 37 or 38 (Wang et al., 2021).

There are some special fields for private transactions in Quorum. These fields are used to make private transactions (Mera, 2019).

- privateFrom: The base64-encoded public key of who sent a private transaction, if the transaction is private

- privateFor / mandatoryFor: if the transaction is private, this field is an array of the base64-encoded public keys of the recipients

- privacyFlag: A number is for the type of private transactions such as Standart Private (0), Counter-Party Protection (1), Mandatory Party Protection (2), or Private State Validation (3)

3.1.4.1 Public Transactions

Public transactions in Quorum are actually normal transactions in Ethereum. These transactions are stored and processed by all nodes in the network. All details of public transactions are visible to all nodes in the network. The term "public" in its name means that it is a global transaction for all nodes in a closed network. There are no such public transactions in the Ethereum network (Baliga et al., 2018).

If a smart contract is to be executed with public transactions, the code of that smart contract must be deployed to all nodes in the network with a public transaction (Baliga et al., 2018).

3.1.4.2 Private Transactions

Private transactions are one of the most important features of Quorum. They provide privacy within the Quorum network. Private transactions work differently than public transactions and go through different transactions (Baliga et al., 2018).

Using private transactions can restrict the sharing of information between nodes in the Quorum network. Private transactions specify which nodes can see and execute that transaction. Other nodes can see that such a transaction has been executed, but they have no information about its contents. Thus, subnets can be created within the network (Mazzoni et al. 2022).

A smart contract loaded with a private transaction is stored only by the nodes specified in the private transaction. Therefore, these private smart contracts are only available through private transactions. The smart contracts that can be accessed between nodes therefore also differ. Tessera is used in Quorum to perform private transactions. Private transactions can be processed and forwarded via Tessera nodes (Baliga et al., 2018).

3.1.5 Network

The Quorum structure is a closed and permissioned blockchain structure. Therefore, only nodes with permission can join the network. To participate in the P2P network, nodes must identify themselves. Each node has its own key. With this key, they can join a network by providing their IP addresses and connection ports (Mazzoni et al. 2022).

The permission system can be further extended, in particular a smart contract-based permission configuration is possible within the Quorum network. This particular permission system can be used to maintain hierarchy within enterprise solutions. Thus, subnets can be created and authorized within a blockchain network (Mazzoni et al. 2022).

The following Figure 3.12 shows the high-level architecture of Quorum in 2 main components. The nodes in the Quorum blockchain network essentially consist of three components: Quorum, Transaction Manager, and Enclave. These three components can be grouped under two labels: Quorum Node and Privacy Manager (Mazzoni et al. 2022).

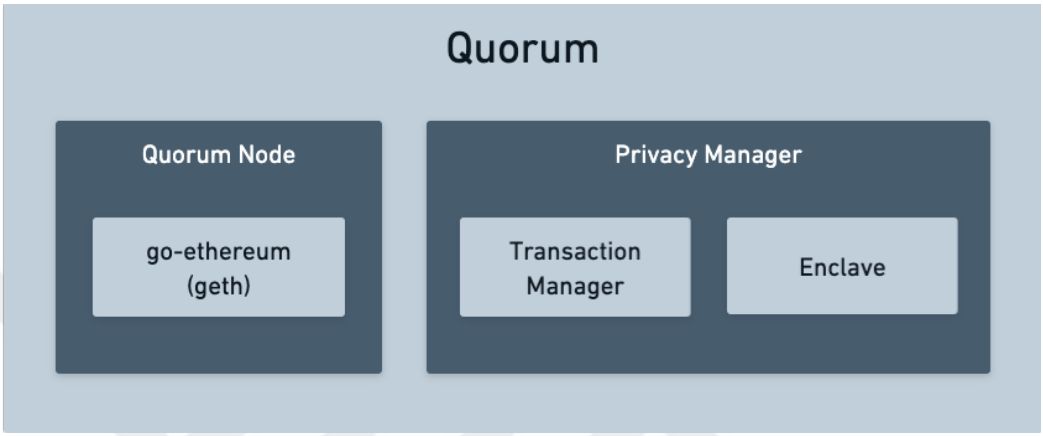


Figure 3.12 High-level architecture of Quorum blockchain

The Privacy Manager contains components that are required for sending private transactions. Since the components in the Privacy Manager are not used in networks that do not require private transactions, it is possible to create a Quorum blockchain network using only the Quorum node component (Mazzoni et al. 2022).

Another name for the Privacy Manager is Tessera node. Tessera is an open-source project and a stateless Java system, which means it does not store any data. The main purpose of Tessera is to perform the cryptographic operations required for the private transaction and is also responsible for distributing the private transactions in the network (Mazzoni et al. 2022).

3.1.5.1 Quorum Node

A Quorum node is basically an adapted form of go ethereum. It is a lighweight version of geth. Classic geth supports only the PoW consensus algorithm, while Quorum node supports consensus protocols such as IBFT, QBFT, or Raft. It has an additional layer of control over the nodes so that a P2P network with approved nodes

can be created. The logic of block creation and validation differs from that of the classic Geth system. It has a global public state root control instead of a global state root control (Baliga et al., 2018).

The storage system in classic Ethereum, the state Patricia Trie, is divided into quorums: the private state Trie and the public state Trie. This allows state changes made with public transactions and private transactions to be stored in two different locations. In this way, the confidentiality of state changes made by private transactions is guaranteed. To support the private transaction structure, the block validation logic also varies, unlike the classic structure. Improvements have been made in the creation of the transaction to ensure that the data in the private transaction reaches all nodes as a hash. The gas fee was reset on the Quorum node. Thus, a gas charge is no longer required for all transactions. The gas consumed by transactions can still be controlled by the gas limit (Baliga et al., 2018).

3.1.5.2 Tessera Node

Quorum uses Tessera as a private transaction manager so that private transactions can be carried out. Tessera is an open-source private transaction manager with Apache 2.0 license. Developed with Java. Quorum and Hyperledger Besu can be used for Ethereum-based enterprise solutions. Its main purpose is to provide a privacy-protected structure for enterprise solutions (Mazzoni et al. 2022).

Tessera can access and store encrypted data in private transactions. Likewise, it can communicate with Tessera nodes of other nodes. Thus, it can distribute private transactions to other Tessera nodes in the network. No private keys are stored within the Tessera node. Instead, the enclave is used for all cryptographic operations. It transmits the keys necessary to decrypt or encrypt the data to the enclave. It has a restful and stateless structure (Mazzoni et al. 2022).

Enclave provides a secure processing environment. It works as a black box and performs all transactions with this principle. In addition, it provides protection against malicious attacks by protecting the information within itself (Mazzoni et al. 2022).

The Enclave is basically responsible for the encryption and decryption operations transmitted to it by the transaction manager. It also provides key management.

Specifically, the tasks of the enclave can be summarized under 2 main headings: Data and Actions (Consensys's Tessera, 2022).

The data-related tasks of the Enclave are as follows:

Private and public key access: Enclave can have a right to reach the private key and the public key of its quorum node

Default public key of nodes: Enclave can access default public keys of all nodes in the network which are stored in the static nodes file.

The tasks of the Enclave under the actions heading are as follows:

1. Reaching the public key of its quorum node

2. Getting public keys of recipient nodes

Encrypting a payload for specific sender and recipients by using the private key of its quorum node

- Encrypting raw payload for a specific sender

- Decrypting payloads that come from other nodes in private transactions

- Adding new recipients for current payloads of transactions

There are two types of enclaves in Tessera: Local enclave and Remote HTTP enclave (Pitkänen, 2021).

Local enclave: This type of enclave operates in the same process as the transaction manager to which it belongs. Here, there is a direct dependency on the transaction manager. However, a logical distinction still exists. The logical behavior of transaction managers and enclave structures is different.

Remote HTTP enclave: Unlike the local enclave, remote HTTP enclave structures have a remote server. Using a server with RESTful endpoints running over HTTP prevents the enclave from running in the same process as the transaction manager. This provides a great separation so that there is no direct dependency between the enclave and the transaction manager. For additional security, the remote HTTP enclave can be deployed to more secure environments. Thus, the process of separating from the transaction manager can be made more secure.

3.2 Keccak Hashing Algorithm

The Keccak algorithm is a hash algorithm. Keccak algorithm became one of the best hash algorithms by winning the SHA-3 Cryptographic Hash Algorithm competition held by the National Institute of Standards and Technology (NIST) in 2012 (Bertoni et al., 2013).

The hash algorithm is a kind of one-way function that returns a fixed-length using data of different lengths. The original data cannot be produced back from the fixed-length data produced. However, the same data always gives the same output, thus leaving a fingerprint. With the fingerprint left, the integrity of the data, that is, whether it has changed or not, becomes understandable.

Hash algorithms are widely used in blockchain structures. For example, Bitcoin uses SHA-256 to monitor the integrity of blocks and transactions. In Ethereum-based structures, the 256-bit Keccak (Keccak-256) hash algorithm is used (Vujičić et al., 2018).

Block headers also contain the hash values of the previous blocks produced with the Keccak-256 hash algorithm. Thus, a change in the blockchain can be easily detected by hash checking. All sent transactions are also signed with the output of the Keccak-256 hash algorithm.

Ethereum addresses are created using the public key. Public keys are used together with the Keccak-256 algorithm to create Ethereum addresses. An Ethereum address consists of 32 bytes, the last 20 bytes of this 32 bytes value are the hash value obtained from the Keccak-256 hash algorithm using the public key (Jezek, 2021). The Keccak-256 algorithm is also used when storing transactions in the Merkle Tree. Thus, the storage cost of data is also reduced (Bertoni et al., 2013; Roussev, 2009).

In this study, the Keccak-256 algorithm is used to create a unique value when requesting access to the data in the IoT device. Likewise, by using the Keccak hash algorithm in the IoT device, transactions are sent to the Quorum blockchain network.

3.3 Recursive Length Prefix

Data serialization is a frequently used method of data transfer. Thus, the data can be moved in certain formats and the transport size can be reduced. The Recursive Length Prefix (RLP) algorithm is an algorithm used when making transaction transfers within the blockchain network in Ethereum-based systems. The transaction to be sent with the RLP algorithm is encoded, then the nodes in the network can simply decode the encoded transactions. The main reason why this algorithm is preferred in the Ethereum structure is that it provides high byte-perfect consistency. In this study, transactions to be signed with the RLP algorithm on ESP32 are encoded (Jezek, 2021).

3.4 Elliptic-Curve Cryptography

Elliptic curve cryptography (ECC) is a method of public-key cryptography based which is on the algebraic structure of elliptic curves over limited fields. On the Quorum blockchain which is Ethereum-based, transactions are signed with this method. Public and private keys of an Ethereum account are used for encrypting. In this study, this cryptography method is used on an ESP32 device within a special purposed ESP Library. The code of ESP32 device can be shown in Appendix 1.

It can provide approximately the same security compared to other encryption methods using keys of the same length. In this way, it needs low memory and low processing capacity. As can be seen in the performance analysis, the lightweight devices are suitable for elliptic curve cryptography.

The discrete logarithm problem on elliptic curves can be explained with defining a elliptic curve, E over a finite field F_q , $q = p^n$ a prime power (Song, 2017).

The set of solutions for an equation is:

$$E\colon y^2 = x^3 + Ax + B,\ (A,\ B \in F_q)$$

That is

$$E(F_q) \coloneqq \{(x,\ y) \in F_q \times F_q \,|\{\, y^2 = x^3 + Ax + B\}.\ E|\}$$

In Quorum, the secp256k1 standard used by Ethereum is used. secp256k1 creates a standard by obtaining the curve required for elliptic curve cryptography with fixed values. It is designed to be 256-bit prime. Elliptic curve secp256k1 is defined as an F_p finite field.

$$p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$

The equation for the elliptic curve, which is defined for faster field arithmetic using a generalized Mersenne prime number, is as follows.

$$E: y^2 = x^3 + 7 \ (A = 3, B = 7)$$

The curve of the standard called secp256k1 is shown in Figure 3.13.

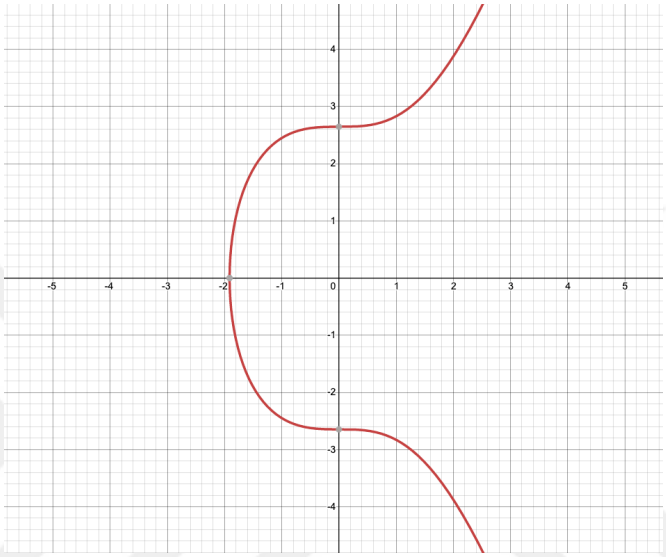


Figure 3.13 Curve of secp256k1 standard

3.4.1 Generating Key Pairs

The range and length of the random number used in the key generation are limited by the curve used. In the secp256k1 algorithm used in Ethereum-based systems, the length of the outlines is determined as 2^{256}, and the range is determined as 0x to 0xFFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFE BAAE DCE9 AF48 AB03 BFD2 5E8C D036 4140 in hexadecimal. The specified range is very wide, it is not possible to reproduce a private key created by brute force. The Bitcoin system also ensures the security of their wallets (Song, 2017).

A production point is determined according to the Curve standard. The private key created with the specified generation point is multiplied, so the point obtained on the curve gives the public key value. This process acts as a one-sided function. Since a

private key generated with a high entropy is used as a multiplier, it is not possible to recalculate or estimate according to the Elliptic Curve Logarithm Problem.

As a result of this algorithm, the public key is in a structure that can be freely shared. Because structurally, it is possible to reproduce a private key with a public key, but on the contrary, it is not possible to obtain a private key with a public key. For this reason, while private key security is a very sensitive issue, especially in public blockchain structures such as Bitcoin, the public key can be shared anonymously with anyone (Avdoshin & Lazarenko, 2018).

3.4.2 Elliptic Curve Digital Signature Algorithm

ECC and Elliptic Curve Digital Signature Algorithm (ECDSA) are structures widely used in blockchain structures as asymmetric cryptology methods. The ECDSA algorithm enables digital signing with public and private keys generated with ECC. In the Ethereum-based Quorum blockchain, transactions are signed using this algorithm. In Ethereum, this signing process is performed for two purposes. The first purpose is to identify the exact account that sent a transaction, and the second purpose is to ensure that the sent transactions are not modified, which can be referred to as storage security (Johnson et al., 2001; Antonopoulos & Wood, 2018).

There are many reasons why the ECDSA algorithm is used in signing transactions. The need for very little computing power can be considered the main reason. In blockchain structures, a highly efficient algorithm that requires little computational power is preferred so that clients who want to interact with a smart contract, especially outside the node, can easily perform their transactions. Another reason is that the generated keys are long enough to be shared. A public key obtained in this way can easily be passed off as an Ethereum or Bitcoin account address (Antonopoulos & Wood, 2018).

In Ethereum-based structures, a transaction must be signed before it is sent to the network. During the signing process, the following values are serialized first:

- nonce: Account’s transaction number for the contract
- gasprice: Gas price for calculating free
- gaslimit: Maximum gas amount to execute the transaction over the contract
- to: Contract address
- value: Cryptocurrency amount, zero for Quorum
- data : The data can be bytecode for deploying a new contract or method and parameters to execute a contract

After these values are serialized using the RLP algorithm, the hash value is obtained using the Keccak-256 algorithm. With the hash value obtained, a signature is created with the ECDSA algorithm using the user's private key value. With this signature, the V, R and S values of the transaction can be sent to the network in hexadecimal form (Hoseini, 2018).

$$Signature = ECDSA\big(Keccak256(RLP(nonce, GP, GL, to, Value), PK)\big)$$

Where;

- ECDSA: Signing Function
- Keccak256: Hashing Function
- RLP: Encoding Function
- PK: Private Key
- GP: Gas Price
- GL: Gas Limit

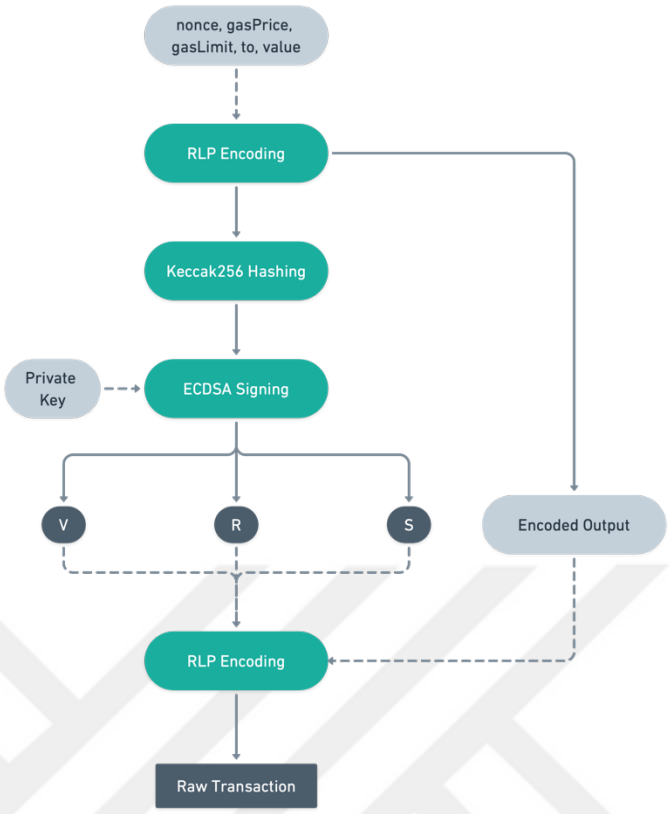


Figure 3.14 Sending raw transaction flow

The generated signature is split into two different parts, depending on the length of the signature, to form the R and S values. The V value, an additional value, allows to decide if it is a private transaction in Quorum, although it is added to speed up the approval process of the sent transaction. If the V value is 37 or 38, the transaction is marked as a private transaction and the fields such as privateFor are checked and the transaction is forwarded to specific nodes. The process of sending raw transactions is shown in Figure 3.14.

3.5 Public Key Recovery

After receiving raw transactions in EVM, the first thing EVM needs to understand is the public key. As you can see from the raw transaction sending process, the sent data does not contain any data of the sender. For this reason, it is necessary to access the sender's public key information to understand the raw transaction sent. The *ECRecover* method is used to perform this transaction in Ethereum.

To use the *ECRecovery* method, you need the hash value generated by the Keccak-256 hash algorithm associated with the transaction and the R and S values from the values formed as a result of the ECDSA signing. Since the purpose of the V value is different, it is not used to calculate the public key in the classical method. In particular, in the Ethereum-based Quorum structure, the purpose of the V-value is used to determine private transactions (Hoseini, 2018).

$$P_k = R^{-1} \times (sR - zG)$$

And

$$P_k = r^{-1} \times (sR' - zG)$$

Where;

- Pk: Public key of sender
- r: r value of signature information
- s: s value of signature information
- R: Ephemeral public key in the ECDSA formula
- z: Lowest n bits of transaction hash
- G: Generator point

As can be seen from the formulas, two different public keys can be obtained by changing the value of R. Which formula to use is decided by whether the V value formed after signing is even or odd. Thus, the public key of the correct sender is calculated. This method is widely used in the Ethereum development environment. There is an *ECRecovery* method in the library that can be used directly in the Solidity programming language, which is especially used to develop Dapps with smart contracts. Accessing the sender address within the smart contract allows many control mechanisms to be added within the smart contract (Hoseini, 2018).

3.6 Elliptic Curve Diffie-Hellman

The Elliptic Curve Diffie-Hellman (ECDH) algorithm is used to generate a key that will be shared between two parties. The two parties can independently choose a common starting point and generate the key required for symmetric encryption. This algorithm was proposed by Neal Koblitz and Victor Miller in the mid-1980s and has become one of the most common key-sharing algorithms over time (Bender & Castagnoli, 1989).

Using this algorithm, two parties can generate a public key using their public keys. The most important point is that both parties can carry out this process without sharing their private keys. Simply, the operation of the ECDH algorithm is described in Figure 3.15. If we determine the names of the two parties as Bob and Alice, by using a common point determined as the Generator Point (G), the parties can create a common secret by performing transactions in turn (Hoseini, 2018).

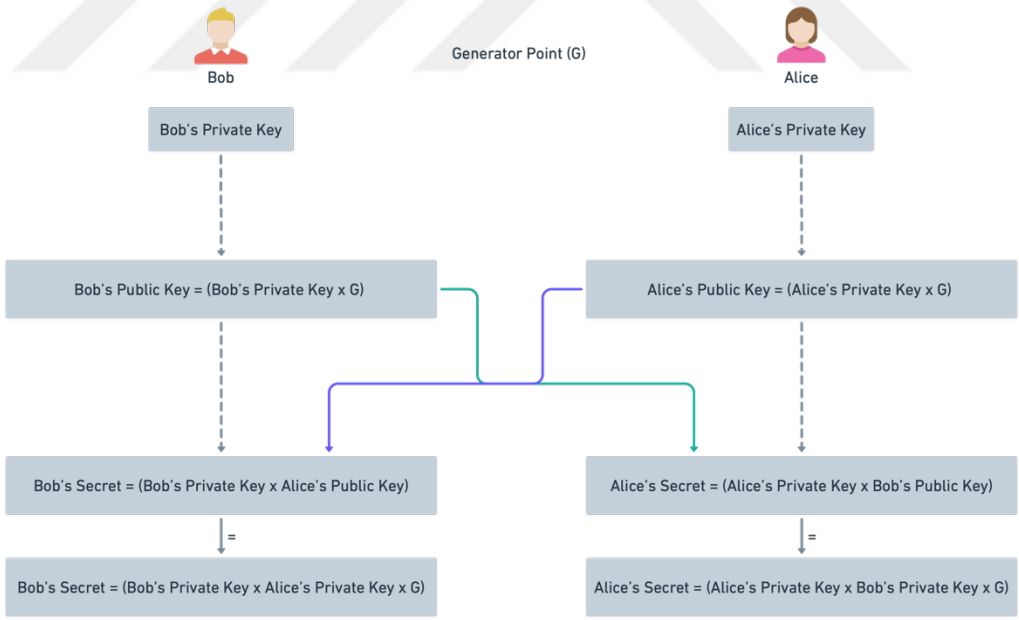


Figure 3.15 ECDH algorithm flow

In this study, the necessary environment for sharing the common starting point required for the ECDH algorithm is used as the blockchain. Thus, the necessary permissions for sharing a common point and for the parties to maintain this communication are securely stored on the blockchain.

CHAPTER FOUR
IMPLEMENTATION

The main goal of this thesis is to securely share data produced on IoT devices. For this purpose, blockchain technology, a distributed and tamper-proof system, has been used. The data produced on the IoT is shared with clients through a control mechanism. Storing all the steps on the blockchain with simultaneous control has made the shared control of data transparent. Things, blockchain and services are used as the development environment. With the services, it is possible for clients to create requests to share.

In this thesis, Quorum blockchain was chosen as the blockchain infrastructure. Quorum is an Ethereum-based blockchain structure. Thus, it contains all the basic features of Ethereum. For this reason, it is possible to use the Ethereum address structure and private key to perform transactions on the blockchain network. The IoT device has private by storing. Thanks to this storage process, the administrators of the network do not have permission or privilege to control the private keys of the devices. Moreover, Quorum has the advantage of being an enterprise blockchain structure. It allows the use of consensus algorithms such as Raft and IBFT. In the implementation of the proposed system in this thesis, the Raft consensus algorithm was preferred as it is more suitable for the development environment.

With the private key stored on the device, it is possible to interact directly with the Quorum blockchain network through the device. By using the ESP32 Web3 library, which was developed specifically for the thesis, IoT devices could interact with a smart contract on the Quorum network and update data directly. The ESP32 Web3 library provides to keep and store the private keys of IoT devices on the device without the need for a specific service.

Since IoT devices are not able to run a Quorum node due to their lack of storage and transaction capacity, they can send raw transactions. With the ESP32 Web3 library, raw transactions can be signed directly with the device's private key.

To store the privacy sharing request, a smart contract published on the Quorum network is used when clients create a request. Thus, all privacy sharing requests are stored in the Quorum blockchain. IoT devices keep track of whether there is a privacy sharing request by sending a request to the smart contract at certain intervals.

The ECDH algorithm is preferred for privacy sharing. The reason why this algorithm is preferred is that it provides high security with less computational power than similar algorithms. Since IoT devices with low capacity also have memory constraints, the ECDH algorithm is very suitable for such devices. Moreover, storing some of the data required for the ECDH algorithm on the blockchain using a smart contract is not a problem. Storing a public key on the blockchain is risky because it can be viewed by other actors on the blockchain network. Instead of storing a public key directly on the blockchain, the initial parameters required for the ECDH algorithm are stored on the blockchain.

The study conducted in this work has shown that it is possible to use a low-capacity IoT device and the Quorum blockchain algorithm, which is an enterprise blockchain structure. No intermediate service is required to use the two together. Since the communication between the IoT device and the blockchain is protected with the ECDSA algorithm, there is no potential security vulnerability. In order for the Quorum blockchain to operate at high performance, the consensus algorithms it contains have ensured that the system operates at sufficient speed.

4.1 Experimental Setup

4.1.1 Quorum Network Specification

The Quorum blockchain network created for the sample implementation in this thesis was created by simulating the network structure with Docker. The specification of the device running Docker is as follows: Intel Core i7 6700HQ 2.6GHZ, 16GB DDR3 RAM, 256GB SSD and GTX950M 4GB. 20.10.14 is used as Docker version. Using Docker Compose on the specified device, a Quorum

blockchain network with 7 Quorum nodes was created. Docker Compose version 1.29.2 is used.

As the consensus algorithm, Raft algorithm was preferred because it is more suitable for testing. The block time is set to 50ms, keeping the default value. All information about the network is defined in the Genesis.json file.

The port information of the Quorum blockchain network is shown in Table 4.1.

Table 4.1 Quorum nodes information

Node Name	RPC Port	Raft Port
Node1	21000	50401
Node2	21001	50402
Node3	21002	50403
Node4	21003	50404
Node5	21004	50405
Node6	21005	50406
Node7	21006	50407

On the side of the ESP32 device, the communication with the Quorum blockchain network was carried out using the 21000 RPC port over Node1.

Due to the Quorum blockchain being a permissioned blockchain, the nodes in the network must be permissioned nodes. Each node must have its unique key. For this reason, in the permissioned-nodes.json file, the key that belongs to each node is determined when creating the network.

4.1.2 Smart Contract Specification

The smart contract developed for this thesis was developed with the Solidity domain-specific programming language. The Solidity version range used for the smart contract has been determined as 0.7.0 to 0.9.0. A simple authorization mechanism has also been created by using the functions in the smart contract. Within this authorization mechanism, an admin address has been determined. The admin address, as the account deploying the contract, is determined during the issuance of

the smart contract and can only be determined later by the account belonging to the admin address.

Many operations can be done with the admin account. The most important of these transactions is the recording of company addresses and addresses belonging to IoT devices on the smart contract. Certain functions can be called with the accounts belonging to the addresses of registered companies. These functions cannot be called even by the admin account. Likewise, some special functions can only be called by accounts belonging to registered IoT devices. As a result of this simple control mechanism, authorization has been made at the smart contract level. Thanks to the authorization, all authorities, including the administrator of the process, and the Quorum blockchain network, cannot interfere with the processes in the smart contract.

Another feature of the smart contract is that all steps in the privacy sharing process are stored in the blockchain network as a single transaction. It is kept tamperproof on the blockchain, without the need to store logs for each step in the sharing process.

The methods in the smart contract are as follows;

- **adminAddress():** It is a method that returns the admin address.

- **approvePrivacySharingRequest(string):** The method that allows the privacy sharing request to be approved.

- **cancelPrivacySharingRequest():** The method that cancels the privacy sharing request.

- **checkIoTDeviceStatus():** A method that queries whether the IoT device is registered.

- **checkPrivacySharingRequest():** A method that queries whether there is a privacy sharing request belonging to the IoT device

- **getIoTPublicKey(address):** The method that calls the public key created after the privacy sharing.

- **registerCompany(address):** A method to register a company.

- **registerIoTDevice(address):** A method to register an IoT device.

- **sendPrivacySharingRequest(address, string, uint256, uint256)**: A method that provides privacy sharing. With this method, a privacy sharing request is created using the parameters. The first parameter is used to specify the IoT device address. The second parameter is the public key of the company which will create the request. The third parameter is used for the nonce value, which is a random number value, and the fourth parameter determines the privacy sharing process in days.

The authorization mechanism in the smart contract is as in Table 4.2.

Table 4.2 Roles and authorization of the smart contract

Method Name	Admin	IoT Device	Company
adminAddress	X	X	X
approvePrivacySharingRequest	-	X	-
cancelPrivacySharingRequest	-	X	-
checkIoTDeviceStatus	X	X	X
checkCompanyStatus	-	-	X
checkPrivacySharingRequest	-	X	-
getIoTPublicKey	-	-	X
registerCompany	X	-	-
registerIoTDevice	X	-	-
sendPrivacySharingRequest	X	-	-

The bytecode required for the publication of the smart contract was compiled by the solidity compiler. The Solidity compiler version used for implementation is 0.8.7.

4.1.3 ESP32 Specification

GetMethodID: This function converts smart contract method names and parameters to hexadecimal and returns a string value of 10 characters. Thus, it is determined which method in a smart contract will be called. Each value to be added to this value represents the parameter of the method. Parameters can be added by encoding.

Call: This function is used to call the return functions in the smart contract after the method name and parameters are determined. It can be called with two additional parameters, the smart contract address and the sender address. The values returned from the smart contract are formatted and returned.

SendTransaction: This function is used to call a method in the smart contract that changes the state. This method creates a transaction. It signs the transaction by taking the nonce value and the sender's private key required to create the transaction. The signed transaction creates an interaction in the smart contract at the address specified.

With these three basic functions, the necessary interaction with a smart contract on the Ethereum-based Quorum network is provided via the ESP32 device. All cryptologic operations required to send a transaction are performed within the Crypto class. Operations such as ECDSA and Keccak256 are handled by native functions in ESP32.

4.2 Proposed Blockchain-Based Privacy Sharing System

This section explains step by step the proposed system to allow sharing of data generated on the IoT device for a certain period. In the proposed system, the Ethereum-based Quorum blockchain structure is used as the blockchain network. The

Quorum blockchain network creates a foundation of trust between IoT devices and companies. A temporary public key is generated to securely share the data generated on the device. Using this key, a secure channel is created between the device and the company.

Before starting the privacy sharing, the environment needs to be prepared and the blockchain network created. In Appendix 2, the Quorum network has been created, given the genesis settings. After the Quorum network is created, the smart contract that will provide the necessary transactions for privacy sharing has been deployed to cover all the nodes in the Quorum network. The example transaction of deploying the smart contract is shown in Appendix 3.

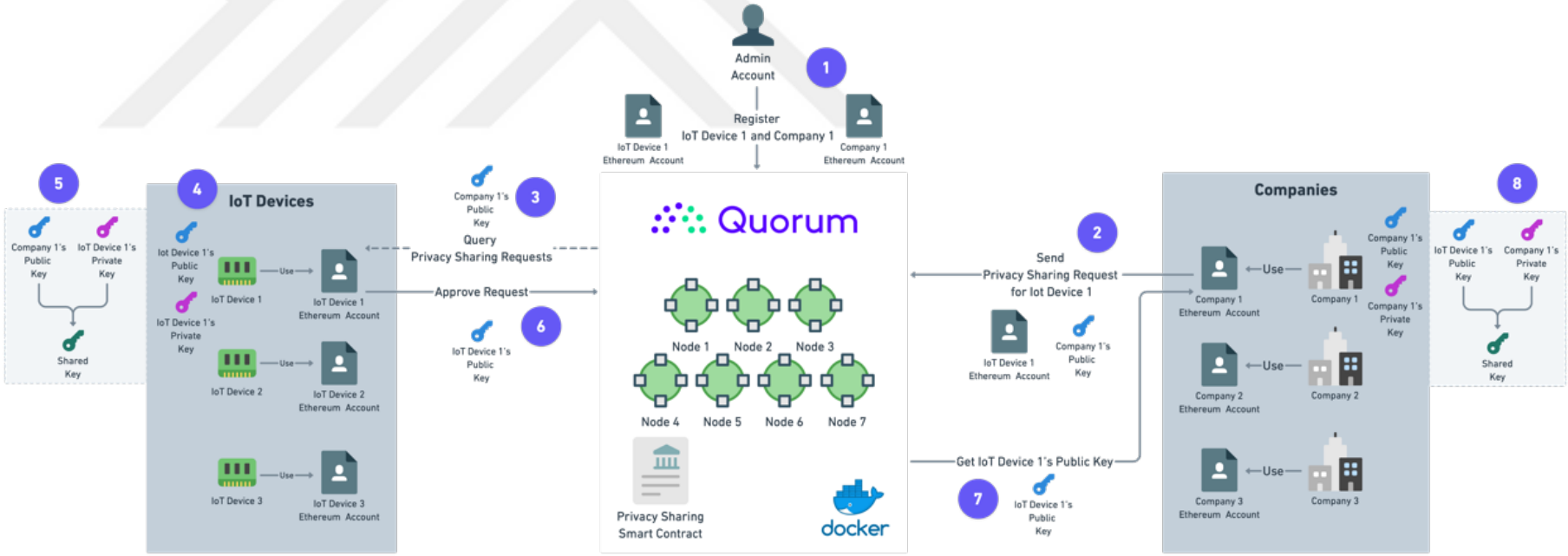


Figure 4.1 Main flow of proposed system

The general flow of the proposed system is shown in Figure 4.1. Quorum blockchain runs on Docker with 7 nodes. The Quorum blockchain has a Privacy Sharing Smart Contract. The admin address is which address published this smart contract. In this flow, general flow is explained in 8 steps:

Step 1: Register companies with After the devices and companies are registered by the admin account with calling the smart contract, the privacy sharing process begins to generate the public key. This sharing process is shown step by step on Figure 4.3. The company creates a public and private key by using the nonce value, which is a random value that it has previously determined, as the generator point.

IoT devices in the system by the admin account using the smart contract.

Step 2: Sending privacy sharing request by the company

Step 3: Querying the privacy sharing request by the IoT device

Step 4: Creating public and private keys on the IoT device

Step 5: Creating a shared key using the company's public key and the IoT device's private key on the IoT device

Step 6: Confirming the privacy sharing request with the public key of the IoT device

Step 7: Getting the public key of the IoT device which approved the privacy sharing request by the company

Step 8: Create the shared key using the company's private key and the IoT device's public key

As seen in the general flow, there are two types of flows in the proposed system. The first flow is the registration of companies and IoT devices to the system. The second flow is the privacy sharing flow. The first flow is shown in Figure 4.2.

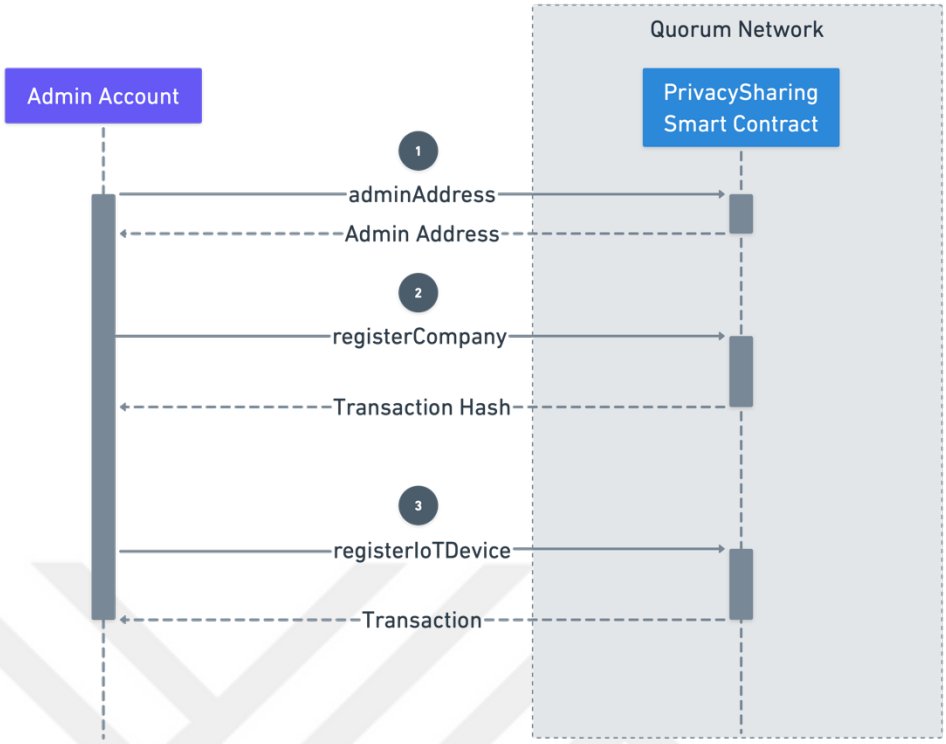


Figure 4.2 Authorization flow via the smart contract

Step 1. After the smart contract is deployed, the admin account calls the adminAccount method in the smart contract. This method returns the admin address information of the smart contract. The return address information and the address information to be processed are compared and confirmed. After the confirmation process is completed, other transactions that can be done with the admin account are started. Since the Transaction Call sent for this transaction is sent only for reading data, it can only be performed using a node in the network without waiting for a response from the leader node in the network. Therefore, it is faster and lower in cost compared to other processes.

Step 2. In order to allow privacy sharing on the smart contract, first of all, the accounts of the companies must be registered in blockchain with the smart contract. The registerCompany method on the smart contract is called by sending the address of the company to be registered as a parameter with the admin account. The transaction created for this request can be seen in Appendix 4. Due to the Raft consensus algorithm which is selected consensus algorithm, the leader node approves the transaction to be added to the blockchain.

Step 3. For IoT devices to call certain methods within the smart contract on the Quorum network, their addresses must be registered. For registering IoT devices on the smart contract, the registerIoTDevice method is called with the address information produced with the private key. This transaction is performed by the admin account, can be seen in Appendix 5. For the transaction got confirmation to be added to the blockchain, the approval of the leader node in the network is awaited. After the leader node approves, the required state change is distributed to the other follower nodes.

After the devices and companies are registered by the admin account with calling the smart contract, the privacy sharing process begins to generate the public key. This sharing process is shown step by step on Figure 4.3. The company creates a public and private key by using the nonce value, which is a random value that it has previously determined, as the generator point.

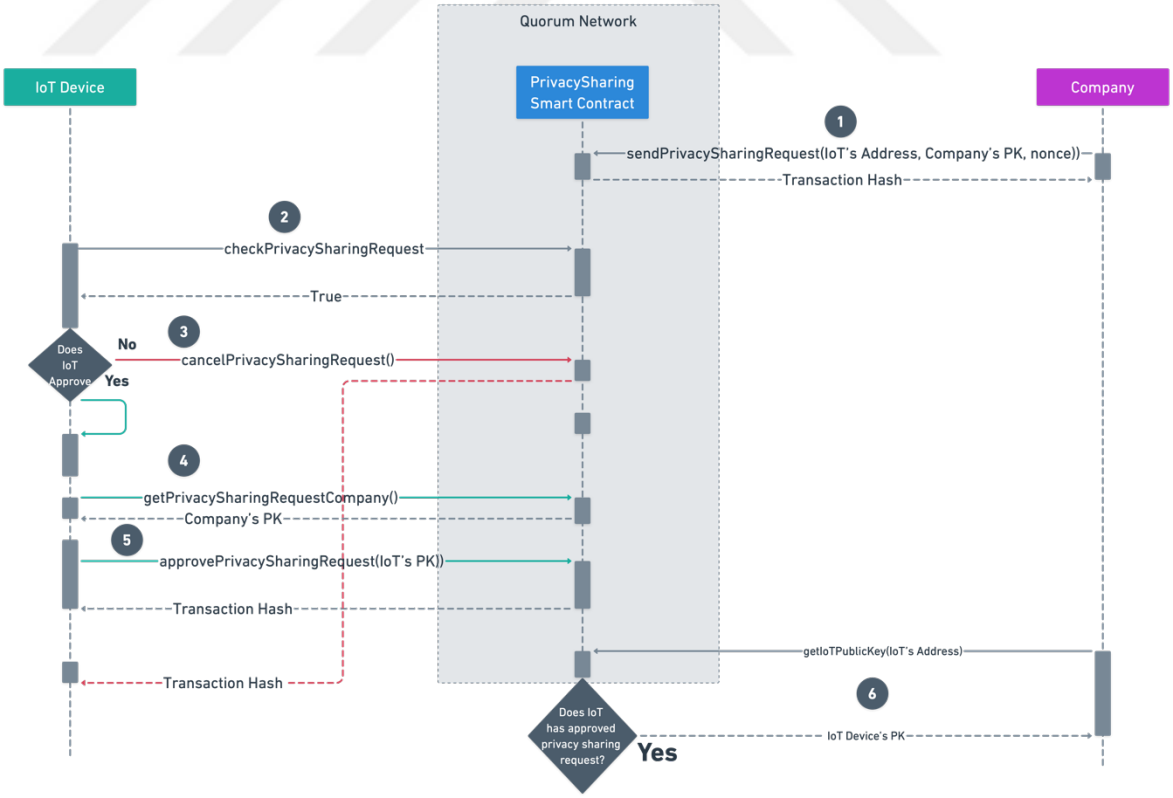


Figure 4.3 Privacy sharing flow

Step 1. The company calls the sendPrivacySharingRequest method on the smart contract using its registered address. When calling the method, it sends the nonce value. After that, the public key is created with the nonce value as a parameter. In addition to this information, it also sends as a parameter from which registered IoT device for how long it wants to create a sharing request. This operation creates a transaction and returns a transaction hash. The approval of the leader node is awaited for this transaction to be confirmed for adding into the blockchain. It waits as a transaction pending transaction until the confirmation comes. An example transaction of the sending privacy sharing request can be seen in Table 4.3.

Table 4.3 Example transaction for privacy sharing request

Transaction Hash	0x40eb047785e85842b96d419e5feb1f682ceee155708408582d7a4eb2bdb40bb0
Status	committed
Block Hash	0x86c17d0199857465d6c0aa07f7cee07495670be20fe9ba7591f3e94c6e66891d
Block Number	4
Gas Used	130988
From	0xd1ff2f4caa91ed568ab353c2b010a322f53cbd34
To	0xe9b0fd81f4ca6629ad1a9e72e08d8914de6b3ec5
Input	0x303de2f3000000000000000000000000000000000000000000000000000000000000000043637300000000000000000000000000000000000000000000000000000000000

Step 2. Every IoT device in the system calls the checkPrivacySharingRequest method on the smart contract with its registered address at certain intervals. This checking call is shown in Table 4.4. This method returns true or false whether there is a pending privacy sharing request for the calling IoT device. If it returns false, nothing is done. If it returns true, the flow continues. The privacy sharing request has reached the IoT device. The IoT device can accept or decline this request. It is possible to make this decision according to an algorithm in the device. However, according to the subject of this thesis, the decision-making algorithm of the IoT device has not been determined for now. If the IoT device declines the request, it goes to Step 3. Otherwise, when the IoT device accepts the request, it goes to the Step 4.

Step 3. In this step, IoT Device calls the cancelPrivacySharingRequest method on the smart contract to reject the request and the flow stops.

Table 4.4 Call example for checking existence of privacy sharing request

Request	{"jsonrpc":"2.0","method":"eth_call","params":[{"from":"0xed9d02e382b34818e88b88a309c7fe71e65f419d","to":"0x248495b7dc58b3d228115e53ce4b48ed9dd85e07","data":"0x130dd733"},"latest"],"id":"0"}
Response	{"jsonrpc":"2.0","id":"0","result":"0x0000000000000000000000000000000000000000000000000000000000000001"}

Step 4. In this step, it can call the getPrivacySharingRequest method on the smart contract to get the privacy sharing request information such as nonce, company's public key and expired date. Then, the IoT device creates a private key and public key pair by using nonce. A shared key is generated by using the IoT device's private key and the company's public key with ECDH algorithm on the IoT device. Creating the shared key with the necessary credentials is shown in Table 4.5.

Step 5. The IoT device calls approvePrivacySharingRequest method on the smart contract with the IoT device's public key as a parameter. In the smart contract, the privacy sharing request is locked and finished. Table 4.6 shows that approval transaction.

Table 4.5 Creating shared key

IoT Device Public Key	CA4CA1B7C188BA824E02FEDE3708A8A28D4237FB915311EBA55F1B86F8B6B13E
IoT Device Private Key	78A1AC6AF22839C90D0E71041A15B220DEDBC782AA2211E41F2DF0C199E2DE4C
Company Key	C8C9746EBEE9F363334639A74CCC88AB1714473FD810E0B94D9C5BBF3AA33002
Shared Key	8512E68FE0A43BF8929C9A5CAA2699FA71577E5A39758F322D96E61FDFD54B22

Step 6. In the last step, the company calls the getIoTPublicKey method on the smart contract, using the ethereum address of the IoT device as a parameter. Thus, it reaches the public key that belongs to the IoT device.

Table 4.6 Privacy sharing confirmation transaction

Hash	71051123bb94060fb4e64b87b3f3e03162eb40e99c610db20a00ec0b1b96cd24
Private Key	e6181caafffff94a09d7e332fc8da9884d99902c7874eb74354bdcadf411929f1
Signature	26216183f16b8941800275a2b156475b089bf0030d7725ac53370c2a4a487e776ad5fb00d8be6a50f76df3401c07fd215b1eadc8197976a9992c4e05c5c9679d
Raw Transaction	f8e52080837a120094248495b7dc58b3d228115e53ce4b48ed9dd85e0780b884df0a011e000000000000000000000000000000000000000000000000000000000000000200000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000404341344341314237433138384241383234453032464544453337303841384132384434323337464239313533313145424135354631423836463842364231334 51ca026216183f16b8941800275a2b156475b089bf0030d7725ac53370c2a4a487e77a06ad5fb00d8be6a50f76df3401c07fd215b1eadc8197976a9992c4e05c5c9679d

After the sharing is done, the company's public key is on the IoT device, while the company's public key is generated by the IoT device. As a result of this sharing, a public key can be generated with the ECDH algorithm by the company. The generated public key can be used for encryption in the next communication. The logs of the process can be inspected in Appendix 6.

CHAPTER FIVE
ANALYSIS OF THE PROPOSED SYSTEM

In this section, the performance analysis of the proposed system and potential security attacks are given in two separate sections. In the last section, the privacy analysis of the proposed system is given.

5.1 Performance Analysis

The main purpose of this study is to share privacy directly between IoT devices and companies using blockchain networks without the need for a third party. For this reason, performance tests were performed on the ESP32-WROOM IoT device selected in this study.

In order to make more consistent performance tests, time measurements were carried out with the *millis()* method belonging to the ESP32 library. This method calculates a time from the moment the device starts to work. Thus, valid time intervals for performance measurements could be calculated.

In this study, since direct communication is established between the ESP32-WROOM device and the Quorum blockchain network, the necessary methods for blockchain communication have been tested.

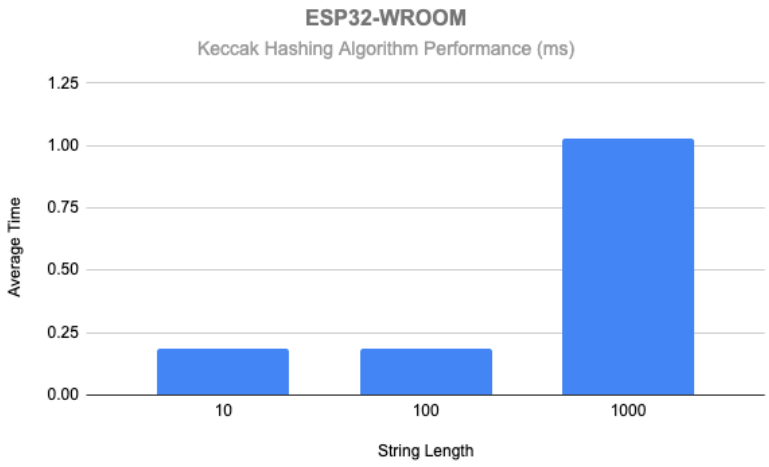


Figure 5.1 Keccak hashing algorithm performance of ESP32-WROOM

Keccak hashing algorithm was tested with three different string lengths as seen in Figure 5.1. Thus, as the length of the data to be hashed increases, it is observed that the processes take longer. The average time, which was 0.18 milliseconds in a 10-character string data, increased to 1.02 milliseconds when the string length was 1000.

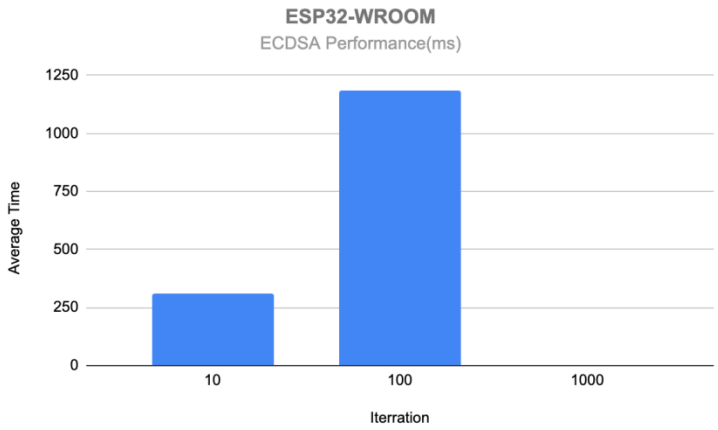


Figure 5.2 ECDSA performance of ESP32-WROOM

The ECDSA performance test used for signing transactions was carried out with three different iterations. As seen in Figure 5.2, the ESP32-WROOM device lost performance as the number of iterations increased. While the number of iterations is 10, the average time is 311 milliseconds. When the number of iterations increased to 100, the average processing time increased to 1182 milliseconds. The tests with 1000 iterations failed and the device rebooted automatically.

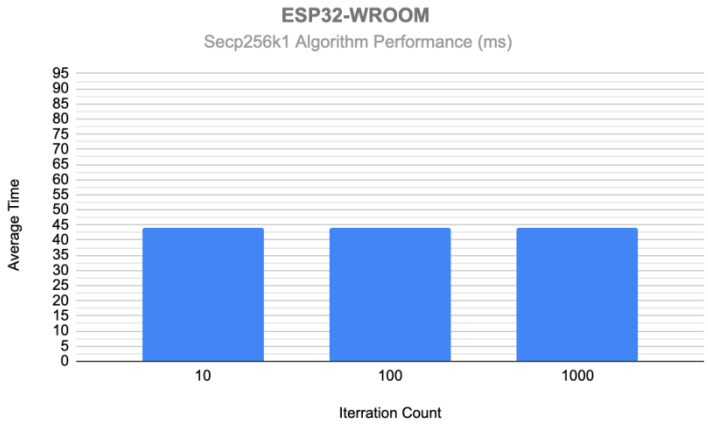


Figure 5.3 Secp256k1 algorithm performance of ESP32-WROOM

As seen in Figure 5.3, the standard Secp256k1 algorithm used to create the elliptic curve did not change, although the number of iterations increased. The ESP32-WROOM device ran the Secp256k1 algorithm perfectly. When the Wi-Fi on the device is turned on and an internet connection is made, the average times increase by about 1 milliseconds as shown in Figure 5.4. ESP32-WROOM board support can be specified as the reason for no change due to iteration. Operations on this device are optimized.

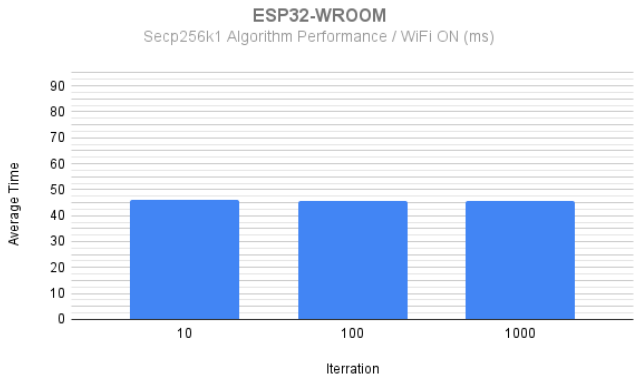


Figure 5.4 Secp256k1 algorithm performance of ESP32-WROOM & WiFi ON

Figure 5.5 shows the performance analysis of the methods developed on ESP32-WROOM. These performance measures are data obtained according to 100 iterations. With the call transactions sent to the Quorum blockchain network, checkIoTDeviceStatus on the smart contract was called, thus questioning whether the IoT device was registered on the smart contract. This polling process took an average of 174 milliseconds. However, it may take about 3 seconds maximum due to the network. The same situation occurred when querying via smart contract whether there is an active privacy sharing request belonging to the device. While this query process takes an average of 119 milliseconds, it can take about 1 second due to the delay in the network.

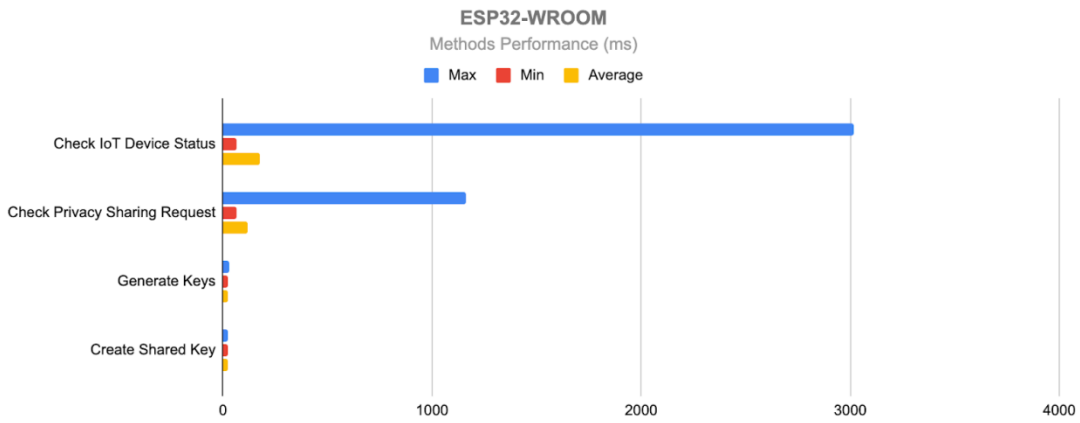


Figure 5.5 Method performance of ESP32-WROOM

During the privacy sharing, temporary public and private key generation and shared key generation took an average of 26 milliseconds. These operations were performed with high performance on the ESP32-WROOM device.

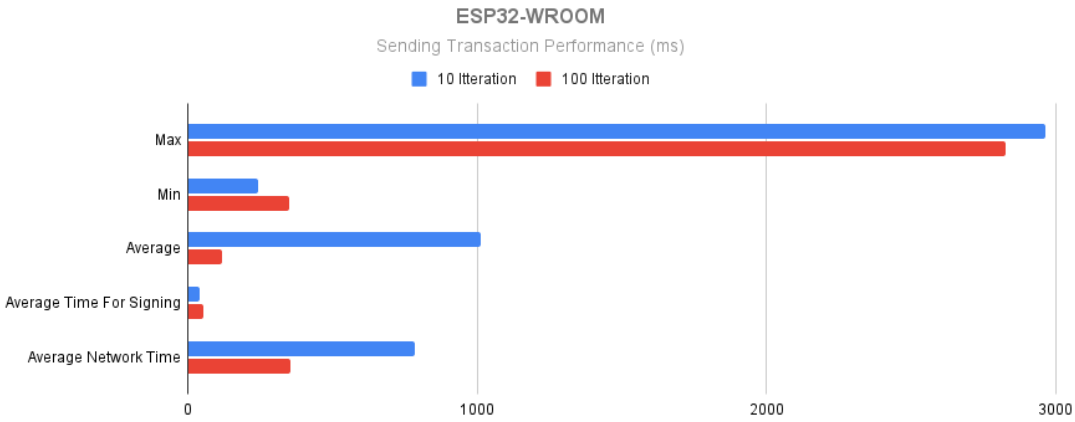


Figure 5.6 Sending transaction performance of ESP32-WROOM

In order for the privacy sharing to be approved by the ESP32-WROOM device, the raw transaction data is signed on the device and sent to the Quorum network. The test of this process was tested with two different iterations. The reason for this is that the consensus algorithm used in the Quorum network is also tested. When 10 iterations were sent, a process took an average of 1 second. However, when 100 iterations are sent, it is seen that the average time drops to 119 milliseconds. Network times decreased from 783 milliseconds to 353 milliseconds when iteration increased from 10 to 100. Since the algorithm used is the Raft consensus algorithm, each block is formed in an average of 50 milliseconds. The effect of this is seen here. While the

transaction signing process on ESP32-WROOM took an average of 42 milliseconds in 10 iterations, it increased to 51 milliseconds in 100 iterations. These changes are shown in Figure 5.6.

5.2 Potential Security Attacks Analysis

5.2.1 Double-Spending

The double-spending problem is one of the most fundamental problems of blockchain systems. In short, it is the transaction that causes the double-spending state change to be processed more than once. In this case, the integrity of the data on the blockchain suffers. The Quorum blockchain structure, which is preferred in the proposed system, uses consensus algorithms to solve the double-spending problem. Consensus algorithms such as Raft and IBFT in the Quorum blockchain provide a fundamental solution to these problems.

In addition, that Quorum is based on Ethereum, the nonce value on the transaction transactions is expected to be unique for each account according to the smart contract it is transacting. On the IoT device, the number of transactions on the smart contract of the account stored on the device is read over the blockchain before privacy sharing is done. Operations are performed by increasing the read nonce value by one. Thus, sending a record with the same nonce value is prevented on the device. After the sent transaction is confirmed, the last nonce value is read over the blockchain before other transactions are made.

5.2.2 Consensus Algorithm Vulnerabilities

The consensus algorithms supported by the Quorum blockchain structure are consensus algorithms suitable for corporate blockchain structures. For this reason, one of the presuppositions of these consensus algorithms is that the nodes in the network are reliable. However, consensus algorithms have some weaknesses of their own.

The Raft algorithm is a voted-based consensus algorithm. The required vote rate to elect a leader node is 50%+1. Considering these conditions, if there is an even number of nodes in the blockchain network, there is a possibility that the leader node will not be selected. If the leader node is not selected, the consensus algorithm cannot work. This poses a potential problem. For this reason, it is recommended not to use the Raft consensus algorithm in production systems.

Due to the vulnerabilities of the raft algorithm, it is more likely for production systems to use the BFT-based IBFT algorithm. However, the BFT algorithm also has some vulnerabilities. One of them is the exponentially increasing message traffic despite the increasing number of nodes. For this reason, it is not recommended to use the BFT algorithm in a blockchain network with a high number of nodes. The IBFT algorithm is an algorithm developed to reduce this exponential increase. It can be used in networks with a high number of nodes, but this algorithm also has a limit.

The Quorum blockchain structure provides a closed and permissioned blockchain infrastructure. All consensus algorithms to be used in this blockchain structure should be chosen correctly according to the usage scenarios. The configurations of the consensus algorithms should also be adjusted according to the usage scenario.

The most important feature of the consensus algorithms in the Quorum blockchain structure is that they can work at high speeds, unlike classical consensus algorithms such as PoW.

5.2.3 Node Capturing

The Quorum blockchain used in the proposed system provides a private and permissioned blockchain network. If a node in the network is malicious or compromised, the administrator of the Quorum network can remove the node from the network.

Nodes in the network can be detected automatically if they perform operations other than the keys defined for them. If some of the nodes in the network maliciously

manipulate the consensus algorithm in the network, they can be easily detected and removed from the network. If a node is compromised, the data may be accessible to the attacker. To prevent this situation, Ethereum addresses used in the network are authorized. Thus, transactions made with addresses that do not have certain authorizations are deemed invalid.

5.2.4 Device Cloning

In the scenario of device cloning, multiple requests can be detected from the same Ethereum address. In order to prevent this situation, the IP address of the device can be determined in the same way and its operations can be restricted according to the IP address.

Specific to the proposed system, it has been seen that such attacks can be prevented with the customizations made on geth if the same device is cloned and a request is sent. The protection of the node in the blockchain network can be customized. RPC requests are restricted according to a certain IP address, so even if the device is cloned, it cannot send an RPC request.

5.2.5 Privacy Keys Security

In this study, the chosen blockchain structure is an Ethereum-based blockchain, transactions are carried out with private keys belonging to Ethereum addresses. Transactions can be signed with the private key stored on the IoT device. The security of the private key on the IoT device is very important. For this reason, it should be stored encrypted on the device. Likewise, the security of private keys belonging to companies is also very important. Private keys in companies are the responsibility of companies.

In case of private keys being compromised, the privileges on the smart contract can be removed by the smart contract administrator. Thus, IoT devices and companies operating on the network become inoperable. With newly defined

Ethereum addresses, reauthorization can be done for IoT devices and companies.

5.2.6 Man-in-the-Middle Attack

In the proposed system, there is no layer between the IoT device and the blockchain. For this reason, IoT devices can communicate directly with the blockchain network. Communication between IoT devices and blockchain is provided by ECDSA. Transactions signed on the IoT device are sent to the Quorum blockchain network.

When an attacker listens to the communication between the IoT device and the blockchain network, the attacker can only see the signed transaction information in an encrypted form. In case of a copy of this data, the attacker can send the signed transaction to the blockchain network multiple times. Due to the fact that the private nonce value of each transaction will remain constant in the copied data, the attacker cannot do anything that will break the data integrity.

During privacy sharing, only the public keys of companies and IoT devices are stored on the blockchain. Private keys created specifically for each sharing request are not transmitted over the network and are used when creating a public key on both sides with the ECDH algorithm. Thus, even someone listening to all communication on the network cannot obtain the generated public key. Listening to all communications in the network is also possible with authorization since a closed and permissioned blockchain network is used.

Besides, smart contracts can be communicated through the RPC ports on each node in the Quorum network. RPC ports can be restricted by IP addresses. In addition to encryption, additional measures can be taken with network management. After determining the IP addresses of the IoT devices and companies, rules can be defined for the nodes in the network.

CHAPTER SIX
CONCLUSION AND FUTURE WORK

In summary, this study examined how data generated on Internet of Things devices can be protected while respecting privacy laws. The important point here is that a device that can communicate directly with the enterprise blockchain structure without a third-party intermediary has temporary shared privacy. Unlike open and permissionless blockchain structures, a blockchain is created in a closed network and a smart contract is developed. Transactions were conducted using more efficient consensus algorithms.

Instead of devices with high hardware capacity such as the Raspberry Pi, devices with lower processing and storage capacity such as the ESP32-WROOM were used. By choosing the ESP32-WROOM as the IoT device, an autonomous key exchange mechanism was developed. It has been shown that it is possible to enable fully autonomous privacy sharing even with a device with low processing capacity.

It has been shown that in the scenario where data generated on IoT devices is shared by enterprises, this privacy sharing can be done temporarily. The ESP32-WROOM device can be used to generate a temporary public key using the Elliptic Curve Diffie-Hellman algorithm. Information such as the duration of the release request can also be transmitted directly to the IoT device via the blockchain. The cryptological algorithms required to perform all of these operations were created on the ESP32-WROOM device. As performance tests on the device show, it takes between 42 and 51 milliseconds to sign a transaction. This time period is quite short for a low-capacity device.

Privacy sharing requests can be stored immutably on the Quorum blockchain. Since it is a closed network structure, the participants of the network can transparently view all transactions. However, they cannot see the created shared key.

In the future, it is possible to develop our own protocol for confirming privacy on the IoT device. In this way, the system can become completely autonomous. A smart contract on the blockchain can be used for this control mechanism.

With the created shared symmetric key, temporary sharing between companies and devices can be organized and provided completely securely. New studies are being conducted to ensure the security of the communication channel here and increase its performance.

REFERENCES

Ahmad, A., Saad, M., Kim, J., Nyang, D., & Mohaisen, D. (2021). Performance evaluation of consensus protocols in blockchain-based audit systems. *2021 International Conference on Information Networking (ICOIN).*

Albrecht, J. P. (2016). How the GDPR will change the world. *Eur. Data Prot. L. Rev., 2*, 287.

Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., Stathakopoulou, C., Vukolić, M., … Yellick, J. (2018). Hyperledger fabric: A distributed operating system for permissioned blockchains. *Proceedings of the Thirteenth EuroSys Conference*.

Antonopoulos, A. M., & Wood, G. (2018). *Mastering ethereum: building smart contracts and dapps.* O'reilly Media.

Avdoshin, S. M., & Lazarenko, A. V. (2018). *Bitcoin Users Deanonimization Methods. Proceedings of the Institute for System Programming of RAS*, *30*(1), 89–102. https://doi.org/10.15514/ispras-2018-30(1)-6

Baliga, A., Subhod, I., Kamat, P., & Chatterjee, S. (2018). *Performance evaluation of the Quorum blockchain platform*. https://doi.org/10.48550/ARXIV.1809.03421

Bender, A., & Castagnoli, G. (2007). On the implementation of elliptic curve cryptosystems. In *Advances in Cryptology — CRYPTO' 89 Proceedings* (pp. 186–192). Springer New York.

Bertoni, G., Daemen, J., Peeters, M., & Van Assche, G. (2013). Keccak. In *Advances in Cryptology – EUROCRYPT 2013* (pp. 313–314). Springer Berlin Heidelberg.

Cachin, C., Anderson, S., An- Droulaki, E., De Caro, A., Christidis, K., Yellick, J., Murthy, C., Nguyen, B., Osborne, M., Schubert, S., Singh, G., Sorniotti, A., Vukolić, M., & Weed, S. (2016). *Architecture of the hyperledger blockchain fabric*.

Clincy, V., & Shahriar, H. (2019). Blockchain development platform comparison. *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*.

Clique. (2022, May 10). *Clique PoA protocol & Rinkeby PoA testnet.* https://github.com/ethereum/EIPs/issues/225

Consensys's Tessera. (2022, May 10). *Enclave types.* https://docs.tessera.consensys.net/en/stable/Concepts/Privacy-Manager/Enclave-types/

Dannen, C. (2017). *Introducing ethereum and solidity: Foundations of cryptocurrency and blockchain programming for beginners.* Apress.

Fullmer, D., & Morse, A. S. (2018). Analysis of difficulty control in bitcoin and proof-of-work blockchains. *2018 IEEE Conference on Decision and Control (CDC)*.

Garewal, K. S. (2020). *Practical blockchains and cryptocurrencies: Speed up your application development process and develop distributed applications with confidence*. Apress.

Goddard, M. (2017). The EU General Data Protection Regulation (GDPR): European Regulation that has a Global Impact. *International Journal of Market Research*, *59*(6), 703–705. https://doi.org/10.2501/ijmr-2017-050

Hamida, E. B., Brousmiche, K. L., Levard, H., & Thea, E. (2017, July). Blockchain for enterprise: overview, opportunities and challenges. *In The Thirteenth International Conference on Wireless and Mobile Communications* (ICWMC 2017).

Hirai, Y. (2017, April). Defining the ethereum virtual machine for interactive theorem provers. *In International Conference on Financial Cryptography and Data Security* (pp. 520-535). Springer, Cham.

Hoseini V. S. (2018). *Mathematics and Data structures in Blockchain & Ethereum*. [Master's Thesis]. University of Turku.

Huang, D., Ma, X., & Zhang, S. (2020). Performance analysis of the raft consensus algorithm for private blockchains. *IEEE Transactions on Systems, Man, and Cybernetics. Systems*, *50*(1), 172–181. https://doi.org/10.1109/tsmc.2019.2895471

Iyer, K., & Dannen, C. (2018). Crypto-economics and game theory. In *Building Games with Ethereum Smart Contracts* (pp. 129–141). Apress.

Jezek, K. (2021). *Ethereum Data Structures*. https://doi.org/10.48550/ARXIV.2108.05513

Johnson, D., Menezes, A., & Vanstone, S. (2001). The elliptic curve digital signature algorithm (ECDSA). *International Journal of Information Security*, *1*(1), 36–63. https://doi.org/10.1007/s102070100002

Kshetri, N. (2017). Can blockchain strengthen the internet of things? *IT Professional*, *19*(4), 68–72. https://doi.org/10.1109/mitp.2017.3051335

Li, S., Xu, L. D., & Zhao, S. (2015). The internet of things: a survey. *Information Systems Frontiers: A Journal of Research and Innovation*, *17*(2), 243–259. https://doi.org/10.1007/s10796-014-9492-7

Liu, Y., Zhang, J., & Zhan, J. (2020). Privacy protection for fog computing and the internet of things data based on blockchain. *Cluster Computing*, *24*(2), 1331–1345. https://doi.org/10.1007/s10586-020-03190-3

Loukil, F., Ghedira-Guegan, C., Boukadi, K., Benharkat, A.-N., & Benkhelifa, E. (2021). Data privacy based on IoT device behavior control using blockchain. *ACM Transactions on Internet Technology*, *21*(1), 1–20. https://doi.org/10.1145/3434776

Mazzoni, M., Corradi, A., & Di Nicola, V. (2022). Performance evaluation of permissioned blockchains for financial applications: The ConsenSys Quorum case study. *Blockchain: Research and Applications*, *3*(1), 100026. https://doi.org/10.1016/j.bcra.2021.100026

Mera, D. (2019). Quorum Blockchain Stress Evaluation in different environments. City University of New York John Jay College of Criminal Justice.

Mohanta, B. K., Jena, D., Ramasubbareddy, S., Daneshmand, M., & Gandomi, A. H. (2021). Addressing security and privacy issues of IoT using blockchain technology. *IEEE Internet of Things Journal*, *8*(2), 881–888. https://doi.org/10.1109/jiot.2020.3008906

Mohanty, D. (2018). *Ethereum for architects and developers: With case studies and code samples in solidity.* Apress.

Mohanty, D. (2019). *R3 Corda for architects and developers: With case studies in finance, insurance, healthcare, travel, telecom, and agriculture.* Apress.

Moniz, H. (2020). *The Istanbul BFT consensus algorithm*. https://doi.org/10.48550/ARXIV.2002.03613

Nadir, R. M. (2019). Comparative study of permissioned blockchain solutions for enterprises. *2019 International Conference on Innovative Computing (ICIC)*.

Nguyen, M. Q., Loghin, D., & Dinh, T. T. A. (2021). *Understanding the scalability of Hyperledger Fabric*. https://doi.org/10.48550/ARXIV.2107.09886

Ongaro, D., & Ousterhout, J. (2014). In search of an understandable consensus algorithm. *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, 305–320.

Pitkänen, E. (2021). *Blockchain in maritime cybersecurity.* [Master's Thesis]. University of Turku.

Polge, J., Robert, J., & Le Traon, Y. (2021). Permissioned blockchain frameworks in the industry: A comparison. *ICT Express*, *7*(2), 229–233. https://doi.org/10.1016/j.icte.2020.09.002

Prusty, N. (2018). *Blockchain for Enterprise: Build scalable blockchain applications with privacy, interoperability, and permissioned features.* Packt Publishing Ltd.

Grammatikis R., P. I., Sarigiannidis, P. G., & Moscholios, I. D. (2019). Securing the Internet of Things: Challenges, threats and solutions. *Internet of Things*, *5*, 41–70. https://doi.org/10.1016/j.iot.2018.11.003

Rocket, Team, Yin, M., Sekniqi, K., van Renesse, R., & Sirer, E. G. (2019). *Scalable and Probabilistic Leaderless BFT Consensus through Metastability*. https://doi.org/10.48550/ARXIV.1906.08936

Roussev, V. (2009). Hashing and data fingerprinting in digital forensics. *IEEE Security & Privacy*, *7*(2), 49–55. https://doi.org/10.1109/msp.2009.40

Schmeelk, S., Rosado, B., & Black, P. E. (2021). Blockchain smart contracts static analysis for software assurance. In *Lecture Notes in Networks and Systems* (pp. 881–890). Springer International Publishing.

Song. J. (2017). *Blockchain 101 - elliptic curve cryptography.* https://eng.paxos.com/blockchain-101-elliptic-curve-cryptography

Srivastava, G., Parizi, R. M., Dehghantanha, A., & Choo, K.-K. R. (2019). Data sharing and privacy for patient IoT devices using blockchain. In *Communications in Computer and Information Science* (pp. 334–348). Springer Singapore.

Szabo, N. (1997). Formalizing and securing relationships on public networks. *First Monday*, *2*(9). https://doi.org/10.5210/fm.v2i9.548

Tanana, D. (2019). Avalanche blockchain protocol for distributed computing security. 2019 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom).

Tikhomirov, S. (2017). Ethereum: state of knowledge and research perspectives. In *International Symposium on Foundations and Practice of Security* (pp. 206-221). Springer, Cham.

Vujicic, D., Jagodic, D., & Randic, S. (2018). Blockchain technology, bitcoin, and Ethereum: A brief overview. *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*.

Wang, W., Wang, L., Xu, S., Wang, J., & Fu, K. (2021). Sharingchain : A privacy protection scheme based on blockchain in the supply chain. *2021 IEEE 4th*

Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC).

Wüst, K., & Gervais, A. (2018). Do you Need a Blockchain? 2018 *Crypto Valley Conference on Blockchain Technology* (CVCBT).

Yamashita, K., Nomura, Y., Zhou, E., Pi, B., & Jun, S. (2019). Potential risks of hyperledger fabric smart contracts. *2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*.

Yang, R., Wakefield, R., Lyu, S., Jayasuriya, S., Han, F., Yi, X., Yang, X., Amarasinghe, G., & Chen, S. (2020). Public and private blockchain in construction business process and information integration. *Automation in Construction*, *118*(103276), 103276. https://doi.org/10.1016/j.autcon.2020.103276

Yu, Y., Li, Y., Tian, J., & Liu, J. (2018). Blockchain-based solutions to security and privacy issues in the internet of things. *IEEE Wireless Communications*, *25*(6), 12–18. https://doi.org/10.1109/mwc.2017.1800116

Yuan, H., Chen, X., Wang, J., Yuan, J., Yan, H., & Susilo, W. (2020). Blockchain-based public auditing and secure deduplication with fair arbitration. *Information Sciences*, *541*, 409–425. https://doi.org/10.1016/j.ins.2020.07.005

Yücedağ, N. (2019). Kişisel verilerin korunması kanunu kapsamında genel ilkeler. *Kişisel Verileri Koruma Dergisi, 1*(1), 47-63. https://dergipark.org.tr/en/pub/kvkd/issue/45759/566993

Zhang, P., Schmidt, D. C., White, J., & Dubey, A. (2019). Consensus mechanisms and information security technologies. In *Advances in Computers* (pp. 181–209). Elsevier.

Zhang, Q., Li, Y., Wang, R., Liu, L., Tan, Y.-A., & Hu, J. (2020). Data security sharing model based on privacy protection for blockchain-enabled industrial Internet of Things. *International Journal of Intelligent Systems*, *int.22293*. https://doi.org/10.1002/int.22293

APPENDICES

Appendix 1: ESP32 Code

crypto.cpp

```
#include "Config.h"
#include "Crypto.h"
#include <string.h>
#include <stdio.h>

#include "mbedtls/aes.h"
#include "mbedtls/md.h"
#include "mbedtls/entropy.h"
#include "mbedtls/ctr_drbg.h"
#include "mbedtls/ecdsa.h"

#include "esp_system.h"
#include "secp256k1_recovery.h"
#include "secp256k1_c.h"
#include "module/recovery/main_impl.h"

void Crypto::ECDSA(uint8_t* hash, uint8_t* sig, int* recid)
{
   secp256k1_context* ctx = secp256k1_context_create(SECP256K1_CONTEXT_SIGN);
   secp256k1_ecdsa_recoverable_signature signature;
   memset(&signature, 0, sizeof(signature));

   if (secp256k1_ecdsa_sign_recoverable(ctx, &signature, hash, Config::PRIVATE_KEY,
secp256k1_nonce_function_rfc6979, NULL) == 0) { return; }

   secp256k1_ecdsa_recoverable_signature_serialize_compact(ctx, sig, recid,
&signature);
}

void Crypto::ECDSA_mbedTLS(uint8_t* hash, uint8_t* sig, int* recid)
{
   mbedtls_ecp_group grp;
   mbedtls_ecp_group_init(&grp);
   mbedtls_ecp_group_load(&grp, MBEDTLS_ECP_DP_SECP256K1);

   mbedtls_mpi d;
   mbedtls_mpi_init(&d);
   mbedtls_mpi_read_binary(&d, Config::PRIVATE_KEY, 32);

   mbedtls_mpi r, s;
   mbedtls_mpi_init(&r);
   mbedtls_mpi_init(&s);

   mbedtls_ecdsa_sign_det(&grp, &r, &s, &d, hash, 32, MBEDTLS_MD_SHA256);
}

uint32_t Crypto::RandomESP()
{
   return esp_random();
}

void Crypto::Random()
{
   mbedtls_ctr_drbg_context ctr_drbg;
   mbedtls_entropy_context entropy;
   uint8_t key[32];

   const char *pers = "aes generate key";
   int ret;

   mbedtls_entropy_init(&entropy);
   mbedtls_ctr_drbg_init(&ctr_drbg);

   if((ret = mbedtls_ctr_drbg_seed(&ctr_drbg, mbedtls_entropy_func, &entropy,
(uint8_t*)pers, strlen(pers))) != 0)
```

Appendix 1: ESP32 Code (Continued)

```
    {
        printf("failed\n ! mbedtls_ctr_drbg_init returned -0x%04x\n", -ret);
        return;
    }

    if((ret = mbedtls_ctr_drbg_random(&ctr_drbg, key, 32)) != 0)
    {
        printf("failed\n ! mbedtls_ctr_drbg_random returned -0x%04x\n", -ret);
        return;
    }
}

void Crypto::SHA3()
{
    const char *payload = "Hello";
    uint8_t hash[32];

    mbedtls_md_context_t ctx;
    mbedtls_md_type_t md_type = MBEDTLS_MD_SHA256;

    const size_t payloadLength = strlen(payload);

    mbedtls_md_init(&ctx);
    mbedtls_md_setup(&ctx, mbedtls_md_info_from_type(md_type), 0);
    mbedtls_md_starts(&ctx);
    mbedtls_md_update(&ctx, (const uint8_t*)payload, payloadLength);
    mbedtls_md_finish(&ctx, hash);
    mbedtls_md_free(&ctx);
}

void Crypto::AES()
{
    uint8_t pt[16] = "Hello World";
    uint8_t ct[16];

    uint8_t key[32];
    uint8_t iv[16];

    memset(key, 0, sizeof(key));
    memset(iv, 0, sizeof(iv));

    mbedtls_aes_context ctx;
    mbedtls_aes_init(&ctx);
    mbedtls_aes_setkey_enc(&ctx, key, 256);
    mbedtls_aes_crypt_cbc(&ctx, MBEDTLS_AES_ENCRYPT, sizeof(pt), iv, pt, ct);

    for (int i = 0; i < 16; i++) { printf("%02x ", ct[i]); }
    printf("\n");

    memset(iv, 0, sizeof(iv));

    mbedtls_aes_crypt_cbc(&ctx, MBEDTLS_AES_DECRYPT, sizeof(ct), iv, ct, pt);
    for (int i = 0; i < 16; i++) { printf("%02x[%c] ", pt[i], pt[i]); }
    printf("\n");

    mbedtls_aes_free(&ctx);
}
```

contract.cpp

```
#include <vector>
#include <cstring>
#include <stdio.h>
#include "Config.h"
#include "Contract.h"
```

Appendix 1: ESP32 Code (Continued)

```
#include "Web3.h"
#include "Helper.h"
#include "Crypto.h"
#include "RLP.h"
#include "TimeTest.h"

using namespace std;

string Contract::GetMethodID(const string& contractName)
{
   string hexContractName = Helper::ByteArrayToHex(contractName);

   string out = Helper::Keccak256(contractName);
   out.resize(10);
   return out;
}


string Contract::Call(const string& data, const string& from, const string& to)
{
   string result = Web3::Call(from, to, data);
   return result;
}

void Contract::SendTransaction(const string& data, uint32_t nonce, const string& to)
{
   // CREATE RLP HEADER
   ContractData contractData;
   contractData.nonce = nonce;
   contractData.to = to;
   contractData.data = data;

   uint8_t signature[SIGNATURE_LENGTH];
   memset(signature, 0, SIGNATURE_LENGTH);
   int recid[1] = { 0 };

   ByteArray encoded;
   ByteArray nonceData = RLP::Encode(Helper::IntToByteArray(contractData.nonce));
   ByteArray gasPriceData =
RLP::Encode(Helper::IntToByteArray(contractData.gasPrice));
   ByteArray gasLimitData =
RLP::Encode(Helper::IntToByteArray(contractData.gasLimit));
   ByteArray toData = RLP::Encode(Helper::HexToByteArray(contractData.to));
   ByteArray valueData = RLP::Encode(Helper::HexToByteArray(contractData.value));
   ByteArray dataData = RLP::Encode(Helper::HexToByteArray(contractData.data));

   encoded.insert(encoded.end(), nonceData.begin(), nonceData.end());
   encoded.insert(encoded.end(), gasPriceData.begin(), gasPriceData.end());
   encoded.insert(encoded.end(), gasLimitData.begin(), gasLimitData.end());
   encoded.insert(encoded.end(), toData.begin(), toData.end());
   encoded.insert(encoded.end(), valueData.begin(), valueData.end());
   encoded.insert(encoded.end(), dataData.begin(), dataData.end());

   // HASH
   ByteArray RLPHeaderData = RLP::EncodeList(encoded);

   string hashedStr = Helper::Keccak256(&RLPHeaderData);


   // SIGN
   ByteArray result = Helper::HexToByteArray(hashedStr);

   uint64_t startTime = TimeTest::GetCurrentMS();
   Crypto::ECDSA(result.data(), signature, recid);
   uint64_t endTime = TimeTest::GetCurrentMS();
   TimeTest::SignTimes.push_back(endTime - startTime);
   // RESULTING RLP DATA
   ByteArray signatureData;
   for (int i = 0; i < SIGNATURE_LENGTH; i++) {
signatureData.push_back(signature[i]); }
   ByteArray R;
```

Appendix 1: ESP32 Code (Continued)

```
 R.insert(R.end(), signatureData.begin(), signatureData.begin() + (SIGNATURE_LENGTH /
2));
   ByteArray S;
   S.insert(S.end(), signatureData.begin() + (SIGNATURE_LENGTH / 2),
signatureData.end());

   ByteArray V;
   V.push_back((uint8_t)(recid[0] + 27));

   ByteArray VData = RLP::Encode(V);
   ByteArray RData = RLP::Encode(R);
   ByteArray SData = RLP::Encode(S);

   encoded.insert(encoded.end(), VData.begin(), VData.end());
   encoded.insert(encoded.end(), RData.begin(), RData.end());
   encoded.insert(encoded.end(), SData.begin(), SData.end());

   ByteArray rawTransaction = RLP::EncodeList(encoded);
   string rawTransactionHex =    Helper::ByteArrayToHex(rawTransaction);

   Web3::SendRawTransaction(rawTransactionHex);
}
```

ecdh.cpp

```
#include "Crypto.h"
#include "Curve25519.h"
#include "ECDH.h"
#include "string.h"
#include <stdio.h>
#include <iostream>
#include "Helper.h"

// For holding ECDH data
static constexpr size_t KEY_SIZE = 32;
uint8_t m_publicKey[KEY_SIZE];          // IoT device private key
uint8_t m_privateKey[KEY_SIZE];         // IoT device public key
uint8_t m_fpublickey[KEY_SIZE];         // The company public key.
uint8_t m_shkey[KEY_SIZE];              // The calculated shared key

void    printHex(const char *label, uint8_t *data, int len) {
   printf("%s:\n", label );
   for ( int i = 0; i < len; i++ ) {
       printf("%02X" , data[i] );
   }
   printf("\n\n");
}

char    highHex( uint8_t v ) {
   char    c;
   v = ( v >> 4 ) & 0x0F;
   v < 10 ? ( c = '0' + v ) : ( c = 'A' + v - 10 );
   return c;
}

char    lowHex( uint8_t v ) {
   char    c;
   v = v & 0x0F;
   v < 10 ?  (c = '0' + v) : (c = 'A' + v - 10);
   return c;
}

// Converts a byte array to an HEX string representing the byte array.
void    Bytes2Str(char *out, uint8_t *key, int len ) {
   int p = 0;
   for ( int i = 0;  i < len ; i ++ ) {
```

Appendix 1: ESP32 Code (Continued)

```
        out[p++] = highHex( key[i] );
        out[p++] = lowHex( key[i] );
    }
    out[p] = '\0';
}

uint8_t nibble( char c )
{
    if ('0' <= c && c <= '9') return (uint8_t)(c - '0');
    if ('A' <= c && c <= 'F') return (uint8_t)(c - 'A' + 10);
    if ('a' <= c && c <= 'f') return (uint8_t)(c - 'a' + 10);
    return 0;
}

uint8_t  getByte( char c1 , char c2 ) {
    return ( nibble(c1) << 4 ) | nibble(c2);
}

// Converts a String with an HEX representation into a byte array
void Str2Bytes( uint8_t *out, char *in , int len) {
    int p = 0;
    for( int i = 0 ; i < len ; i = i + 2 ) {
        out[p++] = getByte( in[i] , in[i+1]);
    }
}

string ECDH::generateKeys() {
    Curve25519::dh1(m_publicKey, m_privateKey);
    // printHex("IoT Device Public Key", m_publicKey, 32);
    // printf("\n\n");
    // printHex("IoT Device Private Key", m_privateKey, 32);
    // printf("\n\n");
    char publicKey[65];
    Bytes2Str(publicKey, m_publicKey, 32);
    return string(publicKey);
}

void ECDH::createSharedKey(char *pubkey) {
    char    s_pubkey[65];
    Bytes2Str( s_pubkey, m_publicKey, KEY_SIZE );
    Str2Bytes(m_fpublickey, (char *)pubkey, 64 );
    printHex("Foreign Key: ", m_fpublickey, 32);
    // Calculate now the shared key
    Curve25519::dh2( m_fpublickey, m_privateKey );
    printHex("Shared Key: ", m_fpublickey , 32);
    memcpy( m_shkey, m_fpublickey, 32 );
}
```

web3.cpp

```
#include "Config.h"
#include <HTTPClient.h>
#include "JSONParser.h"
#include "Web3.h"
#include "TimeTest.h"

string Web3::SHA3(const string& data)
{
    string reqJSON = JSONParser::FormSHA3Request(data);
    string resJSON = Post(reqJSON);
    return JSONParser::GetString(resJSON);
}

uint32_t Web3::GetTransactionCount(const string& address)
{
    string reqJSON = JSONParser::FormGetTransactionCountRequest(address);
    string resJSON = Post(reqJSON);
    return (uint32_t)JSONParser::GetNumber(resJSON);
}
```

Appendix 1: ESP32 Code (Continued)

```
string Web3::SendRawTransaction(const string& data)
{
   string reqJSON = JSONParser::FormSendRawTransactionRequest(data);
   string resJSON = Post(reqJSON);
   return JSONParser::GetString(resJSON);
}

string Web3::Call(const string& from, const string& to, const string& data)
{
   string reqJSON = JSONParser::FormCallRequest(from, to, data);
   string resJSON = Post(reqJSON);
   return JSONParser::GetString(resJSON);
}
string Web3::Post(const string& data)
{
   long startTime = TimeTest::GetCurrentMS();
   HTTPClient client;
   string result;
   client.begin(Config::HOST.c_str(), Config::PORT);
   client.addHeader("Content-Type", "application/json");
   client.addHeader("Connection", "close");
   int responseCode = client.POST((uint8_t*)data.c_str(), data.size());
   string payload = string(client.getString().c_str());
   client.end();
   return payload;
}
```

main.cpp / loop

```
void loop() {
   delay(5000);
   try
   {
       if (checkIoTDeviceStatus())
            {
                if(checkPrivacySharingRequest()){
                 string companyPublicKey = getPrivacySharingRequestCompanyPublicKey();
                 string iotPublicKey = ECDH::generateKeys();
                 ECDH::createSharedKey(strdup(companyPublicKey.c_str()));
                 printf("\n--- IoT PUBLIC KEY---\n%s\n", iotPublicKey.c_str());
                 approvePrivacySharingRequest(iotPublicKey);
                }     else {
                 printf("\n--- There is no Privacy Sharing Request --- \n");
                }
            }
   }
   catch(const std::exception& e)
   {
       std::cerr << e.what() << '\n';
   }
}
```

Appendix 2: Genesis File for Quorum Blockchain with Raft Consensus


```
{
 "alloc": {
   "0xca843569e3427144cead5e4d5999a3d0ccf92b8e": {
     "balance": "1000000000000000000000000000"
   },
   "0x0fbdc686b912d7722dc86510934589e0aaf3b55a": {
     "balance": "1000000000000000000000000000"
   },
   "0x9186eb3d20cbd1f5f992a950d808c4495153abd5": {
     "balance": "1000000000000000000000000000"
   },
   "0x0638e1574728b6d862dd5d3a3e0942c3be47d996": {
     "balance": "1000000000000000000000000000"
   }
 },
 "coinbase": "0x0000000000000000000000000000000000000000",
 "config": {
   "homesteadBlock": 0,
   "byzantiumBlock": 0,
   "constantinopleBlock": 0,
   "petersburgBlock": 0,
   "istanbulBlock": 0,
   "chainId": 10,
   "eip150Block": 0,
   "eip155Block": 0,
   "eip150Hash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
   "eip158Block": 0,
   "qip714Block": 20,
   "isQuorum": true,
   "privacyPrecompileBlock": 0,
   "maxCodeSizeConfig" : [
     {
       "block" : 0,
       "size" : 64
     }
   ]
 },
 "difficulty": "0x0",
 "extraData":
"0x0000000000000000000000000000000000000000000000000000000000000000",
 "gasLimit": "0xE0000000",
 "mixhash":
"0x00000000000000000000000000000000000000647572616c65787365646c6578",
 "nonce": "0x0",
 "parentHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
 "timestamp": "0x00"
}
```

Appendix 3: Smart Contract

```
pragma solidity >=0.7.0 <0.9.0;

contract PrivacySharing {

	address public adminAddress;
	mapping(address => bool) RegisteredCompanies;
	mapping(address => bool) RegisteredIoTDevices;

	struct PrivacySharingRequest {
		uint nonce;
		string companyPublicKey;
		address companyAddress;
		uint expiredDate;
		uint16 status;
		string iotPublicKey;
	}

	PrivacySharingRequest[] PrivacySharingRequestList;
	mapping(address => PrivacySharingRequest) IoTDevicePrivacySharingMap;

	constructor(){
		adminAddress = msg.sender;
	}

	function registerCompany(address _companyAddress) public {
		require(msg.sender == adminAddress, "Caller is not Admin");

		RegisteredCompanies[_companyAddress] = true;
	}

	function registerIoTDevice(address _iotDeviceAddress) public {
		require(msg.sender == adminAddress, "Caller is not Admin");

		RegisteredIoTDevices[_iotDeviceAddress] = true;
	}

	function sendPrivacySharingRequest(address _iotDeviceAddress, string
	memory _companyPublicKey, uint _nonce, uint _dayCount) public {
		require(RegisteredCompanies[msg.sender], "Caller is not
	Company");

		IoTDevicePrivacySharingMap[_iotDeviceAddress] =
		PrivacySharingRequest(_nonce, _companyPublicKey, msg.sender,
		(block.timestamp + (_dayCount * 1 days) ), 1, "");
	}
```

Appendix 3: Smart Contract (Continued)

```
function checkPrivacySharingRequest() public view returns (uint16) {
       require(RegisteredIoTDevices[msg.sender] == true, "Caller is
       not Registered IoT Device");
       require(IoTDevicePrivacySharingMap[msg.sender].nonce != 0 ,
       "Registered IoT Device does not have any request");
       return IoTDevicePrivacySharingMap[msg.sender].status;
}


function getPrivacySharingRequest() public view returns
(PrivacySharingRequest memory) {
       require(RegisteredIoTDevices[msg.sender] == true, "Caller is
       not Registered IoT Device");
       require(IoTDevicePrivacySharingMap[msg.sender].nonce != 0 ,
       "Registered IoT Device does not have any request");
       return IoTDevicePrivacySharingMap[msg.sender];
}

function getPrivacySharingRequestCompanyPublicKey() public view
returns (string memory) {
       require(RegisteredIoTDevices[msg.sender] == true, "Caller is
       not Registered IoT Device");
       require(IoTDevicePrivacySharingMap[msg.sender].nonce != 0 ,
       "Registered IoT Device does not have any request");
       return
       IoTDevicePrivacySharingMap[msg.sender].companyPublicKey;
}

function getPrivacySharingRequestExpiredDate() public view returns
(uint) {
       require(RegisteredIoTDevices[msg.sender] == true, "Caller is
       not Registered IoT Device");
       require(IoTDevicePrivacySharingMap[msg.sender].nonce != 0 ,
       "Registered IoT Device does not have any request");
       return IoTDevicePrivacySharingMap[msg.sender].expiredDate;
}

function getPrivacySharingRequestNonce() public view returns (uint) {
       require(RegisteredIoTDevices[msg.sender] == true, "Caller is
       not Registered IoT Device");
       require(IoTDevicePrivacySharingMap[msg.sender].nonce != 0 ,
       "Registered IoT Device does not have any request");
       return IoTDevicePrivacySharingMap[msg.sender].nonce;
}
```

Appendix 3: Smart Contract (Continued)

```
	function checkIoTDeviceStatus() public view returns (bool) {
		return RegisteredIoTDevices[msg.sender];
	}

	function checkCompanyStatus() public view returns (bool) {
		return RegisteredCompanies[msg.sender];
	}

	function approvePrivacySharingRequest(string memory _iotPublicKey)
	public {
		require(RegisteredIoTDevices[msg.sender], "Caller is not
		Registered IoT Device");
		require(IoTDevicePrivacySharingMap[msg.sender].status == 1 ,
		"IoT Device does not have any request");

		IoTDevicePrivacySharingMap[msg.sender].iotPublicKey =
		_iotPublicKey;
		IoTDevicePrivacySharingMap[msg.sender].status = 2;
	}

	function cancelPrivacySharingRequest() public {
		require(RegisteredIoTDevices[msg.sender], "Caller is not
		Registered IoT Device");
		require(IoTDevicePrivacySharingMap[msg.sender].status == 1 ,
		"IoT Device does not have any request");

		IoTDevicePrivacySharingMap[msg.sender].status = 3;
	}

	function getIotPublicKey(address _iotDeviceAddress) public view
	returns (string memory) {
		require(RegisteredCompanies[msg.sender], "Caller is not
		Company");
		require(IoTDevicePrivacySharingMap[_iotDeviceAddress].status
		== 2, "Request is not approved");
		require(IoTDevicePrivacySharingMap[_iotDeviceAddress].companyA
		ddress == msg.sender, "Company is not matched");
		return
		IoTDevicePrivacySharingMap[_iotDeviceAddress].iotPublicKey;
	}
}
```

Appendix 4: Register Company Transaction

Transaction Property	Transaction Property Value
ID	0x1a2910016f17225d6cbb9b7eb8d5a3cd7db89dd167742625b5d3ef4fdd891edf
Status	committed
Block ID	0xaa8aa039a82643c47c37c36d75da8810187fb7abd63eb95b7b5bd55f0497317a
Block Number	2
Transaction Index	0
Gas Used	43733
Cumulative Gas Used	43733

Transaction inputs & parameters

Return Code	0x1
From	0x0ecb8fab1bdb1243842dff435daabd81ec3d8584
To	0xe9b0fd81f4ca6629ad1a9e72e08d8914de6b3ec5
Value	0
Input	0xa5fef05700000000000000000000000d1ff2f4caa91ed568ab353c2b010a322f53cbd34
Logs	[]
Gas	8000000
Gas Price	0
Nonce	0x1
R	0x63791ac0bbcf50de10f6483ba4c8276066bb725e0f95734eb492fe5ed3d652d7
S	0x265358d518e6abc43085dda8ecb835673260708e3312121f6c8e58873098ad12
V	0x37
Private	False

Appendix 5: Register IoT Device Transaction

Transaction Property	Transaction Property Value
ID	0x1ce25961237f80fa697d4699e3ba90e3e098b281e76a0a4067041e40cb332837
Status	committed
Block ID	0x0a0574b4cd97fe06c01a9464970bd14539cf425f966400e2f5ecffd6e01c4f5f
Block Number	3
Transaction Index	0
Gas Used	43756
Cumulative Gas Used	43756

Transaction inputs & parameters	
Return Code	0x1
From	0x0ecb8fab1bdb1243842dff435daabd81ec3d8584
To	0xe9b0fd81f4ca6629ad1a9e72e08d8914de6b3ec5
Value	0
Input	0x3fd7b41300000000000000000000000ed9d02e382b34818e88b88a309c7fe71e65f419d
Logs	[]
Gas	8000000
Gas Price	0
Nonce	0x2
R	0x60de0eaf59f16e7df217f9b280b758e1640cd9adf0d767303472007182d1e0
S	0x513237662a090040710672cc2944d27c401aec4036c43abe5a90681fc0b51752
V	0x38
Private	false

Appendix 6: Privacy Sharing Process ESP32-WROOM Logs

--- checkIoTDeviceStatus ---
0x0000000000000000000000000000000000000000000000000000000000000001

----- REQUEST -----
{"jsonrpc":"2.0","method":"web3_sha3","params":["0x636865636b5072697661637953686172696e67526571756573742829"],"id":"0"

----- RESPONSE -----
CODE: 200
{"jsonrpc":"2.0","id":"0","result":"0x130dd733708c803bd86676bfeb03a7f50bc5e8952c63bdb4e3b4fce2dcbc7791"}

----- REQUEST -----
{"jsonrpc":"2.0","method":"eth_call","params":[{"from":"0xed9d02e382b34818e88b88a309c7fe71e65f419d","to":"0x248495b7dc58b3d228115e53ce4b48ed9dd85e07","data":"0x130dd733"},"latest"],"id":"0"}

----- RESPONSE -----
CODE: 200
{"jsonrpc":"2.0","id":"0","result":"0x0000000000000000000000000000000000000000000000000000000000000001"}

{"jsonrpc":"2.0","id":"0","result":"0x0000000000000000000000000000000000000000000000000000000000000001"}

--- checkPrivacySharingRequest ---
0x0000000000000000000000000000000000000000000000000000000000000001

----- REQUEST -----
{"jsonrpc":"2.0","method":"web3_sha3","params":["0x67657450726976616379536861726 96e675265717565737443 6f6d70616e795075626c69634b65792829"],"id":"0"}

----- RESPONSE -----
CODE: 200
{"jsonrpc":"2.0","id":"0","result":"0xbcdb02a49e51684d6de437ea9dda3343583488319a472d1722b54244f32de1f0"}

----- REQUEST -----
{"jsonrpc":"2.0","method":"eth_call","params":[{"from":"0xed9d02e382b34818e88b88a309c7fe71e65f419d","to":"0x248495b7dc58b3d228115e53ce4b48ed9dd85e07","data":"0xbcdb02a4"},"latest"],"id":"0"}

----- RESPONSE -----
CODE: 200
{"jsonrpc":"2.0","id":"0","result":"0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000040433843393734364542454539463333633333343633394137344343433838414231373134343733464438313045304239344439433542424633414133333303032"}

{"jsonrpc":"2.0","id":"0","result":"0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000040433843393734364542454539463333633333343633394137344343433838414231373134343733464438313045304239344439433542424633414133333303032"}

--- getPrivacySharingRequestCompanyPublicKey ---
0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000040433843393734364542454539463336333333343633394137344343433838414231373134343733464438313045304239344439433542424633414133333303032

--- getPrivacySharingRequestCompanyPublicKey PUBLIC KEY---
C8C9746EBEE9F363334639A74CCC88AB1714473FD810E0B94D9C5BBF3AA33002
IoT Device Public Key:
CA4CA1B7C188BA824E02FEDE3708A8A28D4237FB915311EBA55F1B86F8B6B13E

IoT Device Private Key:
78A1AC6AF22839C90D0E71041A15B220DEDBC782AA2211E41F2DF0C199E2DE4C

Foreign Key:
C8C9746EBEE9F363334639A74CCC88AB1714473FD810E0B94D9C5BBF3AA33002

Shared Key:
8512E68FE0A43BF8929C9A5CAA2699FA71577E5A39758F322D96E61FDFD54B22
--- IoT PUBLIC KEY---

Appendix 6: Privacy Sharing Process ESP32-WROOM Logs (Continued)

CA4CA1B7C188BA824E02FEDE3708A8A28D4237FB915311EBA55F1B86F8B6B13E

----- REQUEST -----
{"jsonrpc":"2.0","method":"eth_getTransactionCount","params":["0xed9d02e382b34818e88b88a309c7fe71e65f419d","latest"],"id":"0"}

----- RESPONSE -----
CODE: 200
{"jsonrpc":"2.0","id":"0","result":"0x20"}

----- REQUEST -----
{"jsonrpc":"2.0","method":"web3_sha3","params":["0x617070726f6655072697661637953686172696e6752657175657374287374726965729"],"id":"0"}

----- RESPONSE -----
CODE: 200
{"jsonrpc":"2.0","id":"0","result":"0xdf0a011e172db71951febfc06cf4254ebb849143db19579b7120a24efda3ca50"}

CONTRACT DATA: -
*0xdf0a011e000000000000000000000000000000000000000000000000000000000000000000000000000000200000000000000000000000000000000000000000000000000000000000

Appendix 6: Privacy Sharing Process ESP32-WROOM Logs (Continued)

----- REQUEST -----
{"jsonrpc":"2.0","method":"eth_call","params":[{"from":"0xed9d02e382b34818e88b88a309c7fe71e65f419d","to":"0x248495b7dc58b3d228115e53ce4b48ed9dd85e07","data":"0xd380a680"},"latest"],"id":"0"}

----- RESPONSE -----
CODE: 200
{"jsonrpc":"2.0","id":"0","result":"0x0000000000000000000000000000000000000000000000000000000000000001"}
--- checkIoTDeviceStatus ---
0x0000000000000000000000000000000000000000000000000000000000000001

----- REQUEST -----
{"jsonrpc":"2.0","method":"web3_sha3","params":["0x636865636b5072697661637953686172696e67526571756573742829"],"id":"0"}

----- RESPONSE -----
CODE: 200
{"jsonrpc":"2.0","id":"0","result":"0x130dd733708c803bd86676bfeb03a7f50bc5e8952c63bdb4e3b4fce2dcbc7791"}

----- REQUEST -----
{"jsonrpc":"2.0","method":"eth_call","params":[{"from":"0xed9d02e382b34818e88b88a309c7fe71e65f419d","to":"0x248495b7dc58b3d228115e53ce4b48ed9dd85e07","data":"0x130dd733"},"latest"],"id":"0"}

----- RESPONSE -----
CODE: 200
{"jsonrpc":"2.0","id":"0","result":"0x0000000000000000000000000000000000000000000000000000000000000002"}
{"jsonrpc":"2.0","id":"0","result":"0x0000000000000000000000000000000000000000000000000000000000000002"}
--- checkPrivacySharingRequest ---
0x0000000000000000000000000000000000000000000000000000000000000002

--- There is no Privacy Sharing Request ---