DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

BLOCKCHAIN BASED AUTHORIZATION MODEL FOR INTERNET OF THINGS DEVICES

by
Alper ALTUN

October, 2019
İZMİR

BLOCKCHAIN BASED AUTHORIZATION MODEL FOR INTERNET OF THINGS DEVICES

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

Alper ALTUN

October, 2019
İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"BLOCKCHAIN BASED AUTHORIZATION MODEL FOR INTERNET OF THINGS DEVICES"** completed by **ALPER ALTUN** under the 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. Yalcın ÇEBİ

(Jury Member)

Asst. Prof. Dr. Enis Karaarslan

(Jury Member)

Prof. Dr. Kadriye ERTEKİN
Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENTS

I would like to thank my supervisor, Assoc. Prof. Dr. Gökhan DALKILIÇ for his guidance and help throughout this work. I would also like to thank my family, my friends, and all my teachers, who have contributed to me in my education up to present, for their continuous support. Without their encouragement and continuous support, this research would not have been possible.

Alper ALTUN

BLOCKCHAIN BASED AUTHORIZATION MODEL FOR INTERNET OF THINGS DEVICES

ABSTRACT

The Internet of things has many unexplored potentials. Features such as low purchasing costs, low energy consumption, and small size make the internet of things indispensable and irreplaceable. However, the fact that they only have enough processor power to run particular tasks has led to the failure of implementation of many traditional security measures to the IoT environment, which are developed to date. Moreover, the lack of knowledge of developers about the security measures needs to be taken, creates lots of security loopholes and causes unauthorized persons to seize a lot of valuable information which undermines the trust of the users. Likewise, the unpredictable and rapid growth of IoT devices highlighted the importance of scalability, authorization, and encryption issues. Although partial solutions are being suggested to address these concerns, a generally accepted solution still doesn't exist.

Thanks to its tamper-resistant, decentralized, distributed, programmable structure, Blockchain-based authentication, and access control systems can be used to strengthen IoT security. In this study, an application that provides authorization control and communication confidentiality without a reliable third party over the Blockchain network on restricted devices is presented. Furthermore, the security analysis that can be provided with this application is discussed by exemplifying specific attacks and performance data related to the limited device used are presented.

Keywords: ESP32, gateway, internet of things, Blockchain, security, constrained device, authentication, confidentiality

NESNELERİN İNTERNETİ CİHAZLARI İÇİN BLOKZİNCİRİ TABANLI YETKİLENDİRME MODELİ

ÖZ

Nesnelerin İnterneti keşfedilmemiş birçok potansiyele sahiptir. Düşük satın alma maliyetleri, düşük enerji tüketimi ve küçük boyut gibi özellikleri, nesnelerin interneti teknolojisini vazgeçilemez kılmaktadır. Ancak, yalnızca belirli görevleri yerine getirmek için yeterli işlemci gücüne sahip olmaları, bugüne kadar geliştirilen geleneksel güvenlik önleminin nesnelerin interneti için uygulanamamasına neden olmuştur. Dahası, geliştiricilerin güvenlik önlemleri konusundaki bilgi eksikliğine sahip olmaları, birçok güvenlik boşluğu yaratmakta ve yetkisiz kişilerin birçok değerli bilgiyi ele geçirmesine neden olmakta ve bu durum kullanıcıların güvenini zedelemektedir. Benzer şekilde, öngörülemeyen ve hızlı büyüme nedeni ile, ölçeklenebilirlik, yetkilendirme ve şifreleme konularının önemini arttırdı. Bu kaygıları gidermek için kısmi çözümler önerilmiş olmasına rağmen, yaşanılan problemlerin genelini hedef alan bir çözüm hala mevcut değildir.

Kurcalamaya dayanıklı, merkezi olmayan, dağıtılmış, programlanabilir yapısı sayesinde, Blockchain tabanlı kimlik doğrulama ve erişim kontrol sistemleri, IoT güvenliğini güçlendirmek için kullanılabilir.Bu çalışmada, kısıtlı cihazlar üzerinde Blockchain ağı kullanarak güvenilir üçüncü taraf olmadan yetkilendirilmesin ve iletişim gizliliği sağlayan uygulama sunulmuştur. Ayrıca, bu uygulama ile sağlanabilecek güvenlik analizi, belirli saldırılar örneklendirilerek tartışılmış ve kullanılan kısıtlı cihaz ile ilgili performans verilerine yer verilmiştir.

Anahtar kelimeler: ESP32, ağ geçidi, nesnelerin interneti, Blockchain, güvenlik, kıstılı cihazlar, kimlik doğrulama, gizlilik

CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

CHAPTER ONE
INTRODUCTION

Today's industry firms discovered that increasing the productivity of workers, preventing waste, eliminating errors in the production process and ensuring efficiency and increasing profitability with all these steps are possible with the Internet of things technology. The Internet of things (IoT) has been making huge investments by industry firms. Internet of things, which has developed rapidly thanks to the investments made, has also become the most popular technological subject of today with its use in areas such as smart homes, smart cities, smart cars. The *thing* has a comprehensive meaning. To be considered as a *thing* , a device capable of making connections must be uniquely identifiable on the Internet. The term “Internet of things” emerged to reflect the growing number of smart and connected products and highlighted new opportunities they could bring. That is their expanded capabilities and the data they can produce (Michael & James, 2015).

To prevent unauthorized users (people or devices) from accessing the system, data privacy, confidentiality, and integrity, as well as authentication and authorization mechanisms, must be guaranteed. However, concerning the requirement of confidentiality, both the data protection and the privacy of users' personal information must be ensured, as devices can produce sensitive information. Because the IoT environment is characterized by different devices that have to process and use data under user needs and rights, trust is the most fundamental and most crucial issue (Sicari et al., 2015).

When the variety of sensors and the increasing smart devices are considered together, scalability will appear to be a significant problem, especially for large-scale intelligent topologies. Poorly thought-out naming and addressing will ultimately prevent the addition of new objects into the topology and eliminate the topology's manageability.

A Blockchain-based Authentication and Authorization mechanism has been proposed to overcome the problems which are still current in the Internet of things in this thesis. To demonstrate the feasibility of this proposed study, the ESP32 development kit, which has all the qualities an object should possess, was preferred. Besides, the anonymous secret key was created between the parties in contact, and confidentiality was achieved during the communication of these parties on the network. Finally, using the blockchain-based addressing structure, the risk of collusion is virtually eliminated, and the scalability problem has been shown to be overcome.

1.1 Internet of Things

Internet of things, a community of low-power sensors and actuator devices, has many uses today. The most noteworthy of these areas of use are smart cities, smart homes, connected farming, connected health areas (Gour, 2018). Any electronic device with a structure that allows data transfer can be considered a *thing*. The first condition for a device to be referred to as a *thing* in the Internet of things community is that it must be uniquely identified on the network.

For the Internet of things, energy consumption, production cost, security, and data processing are among the most critical issues. A manufactured sensor or actuator should consume as little power as possible and be available at an affordable price. In order to meet this requirement, these *things* are produced with low processing power. Therefore, the protocols and measures used for the traditional network may not be used for the communication of *things*. There are many different views on how many layers of the Internet of things architecture consist. However, as many researchers agree, the Internet of things consists of three layers (Mahmoud et al., 2016; Yang et al., 2011; Zhao & Ge, 2013).

Perception layer: This layer is also known as a layer of sensors. It senses, collects, and processes data from the environment using sensors and actuators. It then sends this information to the Network Layer. The biggest challenge in this layer is to obtain more sensitive and high accuracy data with low power and low cost. Temperature,

Proximity, Pressure, Gyroscope sensors are the most commonly used sensors (Macharla, 2018).

Network layer: In the network layer, the *things* interact with each other or with an IoT gateway. During the interaction, protocols specially developed or optimized for the Internet of things are used. Bluetooth Low Energy (BLE), Z-Wave, ZigBee Smart, 802.11ah, 802.15.4e, and LoRa are examples of developed protocols to meet the Internet of things requirements (Al-Sarawi et al., 2017; Salman & Jain, 2017).

Application layer: The application layer is responsible for providing services that make the data produced on the perception layer accessible on the Internet when requested (Karagiannis et al., 2015). The application layer should also ensure the accuracy, integrity, and confidentiality of the data (Yang et al., 2011). In this layer, two communication patterns are widely used: publish-subscribe and request-response. Message Queuing Telemetry Transport (MQTT) and Constrained Application Protocol (CoAP) are protocols developed for the Internet of things systems and are mainly used in these systems (Asim, 2017). MQTT uses the publish-subscribe communication pattern, while CoAP uses the request-response communication pattern (Naik, 2017).

1.1.1 Application Layer Protocols

1.1.1.1 MQTT

MQTT is one of the oldest Internet of things protocols that provides machine to machine communication. Designed by IBM in 1999 and adopted as a communication standard in 2013. In this protocol, which uses the publish / subscribe communication pattern, each party to contact must be connected to the broker. It uses transmission control protocol / Internet protocol (TCP / IP) on the transport layer. MQTT is binary based protocol, so data needs to be serialized to binary format before sending over the Internet.

The MQTT package transferred over the network consists of two parts. Topic and message, where the topic is used to filter messages for each connected client. Each message is published on an address called a topic.

MQTT supports multi-casting (more than one client can subscribe to a topic). There is only a 2-byte of header information per message. It is possible to send messages up to 256 MB in size. There are three different levels of quality of service in this protocol.

- **QoS 0:** The message is only published. Acknowledgment signal is not expected.

- **QoS 1:** The acknowledgment signal is expected after a particular time after the message is published. If the ACK message is not received or disappears, the client will resend the same message actually received by the broker, which will result in a duplicate message on the broker.

- **QoS 2:** the 4-Way handshake is used to ensure that the message is delivered only once. However, this complex process will extend the data transmission time between the client and the broker.

1.1.1.2 CoAP

It is a machine-to-machine (M2M) communication protocol that is designed by IETF for devices with constrained resources. It uses GET, POST, PUT and DELETE commands, which are also used for the hypertext transfer protocol (HTTP). CoAP uses the request/response, communication model. It uses the user datagram protocol (UDP) on the transport layer.

CoAP protocol supports multi-casting using the UDP protocol used in the transport layer. Each CoAP message has 4-byte header information, and in general, the total size of a message does not exceed an Ethernet frame size, which is 1500 bytes in size.

It uses the request / response model. Uniform resource identifier (URI) is used to address messages, so unlike MQTT, it is not necessary to subscribe to a topic in order to send or receive a message. In micro-services, the CoAP protocol is widely used.

Two levels of Quality of Service (QoS) is supported.

- **Non-Confirmable:** Non-critical messages can be sent with this quality of service level; no acknowledgment is expected from the receiver after the published message (Fire-and-forget).

- **Confirmable:** The acknowledgment is expected for the published message. This information can be sent synchronously, or asynchronously, as a separate message. For the detection of duplicate messages, there is a field called Message-ID.

The CoAP protocol is a subset of the HTTP protocol. Thanks to this feature of the protocol, it is possible to convert and transfer data from this protocol to the HTTP protocol by developing a simple proxy.

1.1.2 Internet of Things Gateway

Gateways can filter and process the data generated by the sensors while meeting the storage and processing power requirements of the sensors. Another purpose of the gateway is protocol translation. That is, it reads the data from the sensors from the mostly short-range protocols designed for IoT and transmits them to the traditional network using traditional transport layer and application layer protocols. In order to achieve this, the gateway device must support many application protocols and many network protocols, including wired and wireless types (Zhong et al., 2016).

Things can generate hundreds of data in seconds. These data should be interpreted and valued. The gateway reduces the traffic generated on the network by preprocessing

the data generated by the *things* (Aazam et al., 2014; Zhong et al., 2016). Another task of the IoT Gateway is to create a security layer. By preventing devices with limited resources from going directly to the internet, a layer of security against attackers is created (McClelland, 2017).

1.1.3 Problems on the Internet of Things

Scalability problems are the first problem that is felt more frequently with the uncontrolled growth of IoT topologies. The ability to uniquely identify then incorporate *things* into the expanding network is one of the major problems of the Internet of things networks (Gupta et al., 2017).

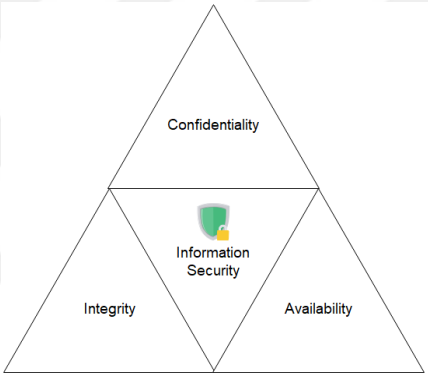

Figure 1.1 Information security requirements triad

The information security requirements of the IoT networks are the same as those of traditional networks. But; many security measures applied to traditional networks cannot be applied to the IoT networks. The lack of standardization of IoT technology is another factor that causes security problems. Information security requirements for any network, including the IoT network, are categorized under three main headings: privacy, availability, and integrity. These three main titles are abbreviated and referred to as the CIA model (Abomhara & Koien, 2014; Vorakulpipat et al., 2018). The common representation of the CIA model is given in Figure 1.1. The CIA model, in which these headings are collected, provides guidance on the security policies that should be involved in a topology (Haughn, Matthew; Giblisco, 2014).

- **Confidentiality:** It aims to prevent unauthorized persons or attackers from accessing the data produced or stored. Authentication and Authorization processes ensure that the data cannot be accessed by unauthorized persons, while the data is encrypted with end-to-end encryption with cryptography algorithms.

- **Integrity:** It is the combination of the measures taken to ensure the consistency, accuracy, and reliability of the data. Data must not be altered during transmission or modified by unauthorized persons from where it is stored. The consistency of the data can be checked with CRC algorithms, and regular backups should be taken to prevent hardware failure. It provides mechanisms of authorization to prevent data from being changed by attackers or unauthorized persons.

- **Availability:** It is the combination of the measures to be taken to ensure the designed topology to operate continuously. Redundancy against hardware failures, failover, disaster recovery against natural disasters, as well as measures to be taken against bottlenecks and DDoS attacks are considered under this heading.

As a result; a strong Authentication and Authorization system is needed to be designed in order to ensure security and scalability in the IoT systems for these systems. Designing an infrastructure that uniquely identifies electronic devices on the network will eliminate the problem of scalability, authentication mechanisms placed in the system will prevent unauthorized persons from accessing the system and thus will provide system confidentiality and additionally, by means of robust authorization mechanisms to be developed, integrity requirements will be met by preventing unauthorized changes to the data contained in the system (Ye et al., 2014).

1.3 Authentication

Authentication is the cornerstone of all communication systems, including IoT systems. It is the key technology used for information security and prevents unauthorized access and modification of data by attackers (Babaeizadeh et al., 2015). Authentication is an essential part of the CIA Triad model used for the development of security mechanisms (Farooq et al., 2015).

1.3.1 Authentication Factors

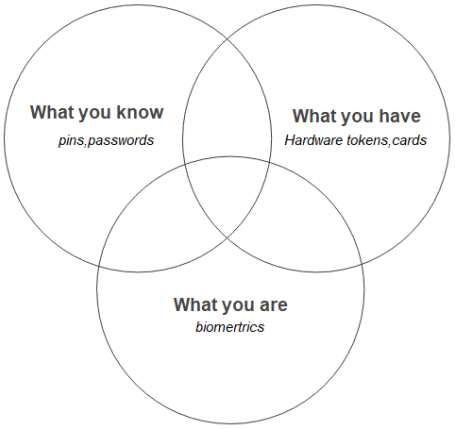

Figure 1.2 Authentication factors

Starting from the introduction of Internet technologies to our lives, a wide variety of authentication techniques have emerged, and many methods have been launched. To fully understand these technologies, the forms on which they are based must be well known. When you claim that you are someone on a target, you must verify that claim. The factor refers to the types of data used in the authentication process. As illustrated in Figure 1.2, there are three generally accepted factors, namely, what you know, what you have, what you are, are all described below. (Dias, 2017; Pandya et al., 2015).

What you know: It is the use of confidential information stored in the memory of the person or device for authentication when necessary. The most crucial point here is that this information should only be known by the person or device concerned.

Information such as a password, pin, or answer to a predefined secret question is an example of "what you know" factor.

What you have: Confidential information is stored in another physical object rather than stored in memory. During the authentication process, this physical item is activated and used to obtain the desired information for the authentication process. Smart cards with integrated circuits and token generators, which are frequently used in banking operations today, are examples of goods.

What you are: It is the use of a feature of the person or device capable of identifying that person or device for authentication. The essential requirement for this feature is that this information must be guaranteed to be unique. Biometrics is the most explicit example for individuals. Fingerprint, handprint, palm, retina are samples of biometric information of a person. For some specific devices, there is a physically unclonable function module that functions as a digital fingerprint. During the production phase of the device, this module is added to the device, and the required parameters are defined as permanent and unchangeable. This module is not included in many development kits because it increases the cost of production (Braeken, 2018; Maes, 2013; Mughal et al., 2018).

1.3.2 Authentication Methods

Password Authentication: Due to its simple structure, it is still the most widely used authentication method in applications today. In this structure, the identity and password of the user are stored in the database by being encrypted. In order to accept this method as secure, it is necessary to use a password only in one system, but also to create a long and complex password (Fenton et al., 2017). In this method, users constitute the weakest link of security (Bonneau et al., 2015). This method is also subject to many malicious attacks such as brute-force and dictionary attack due to the fact that users create weak passwords, and even if they want to create strong passwords, the password that can be created is limited to combinations that can be created by using a keyboard (Yassin et al., 2012).

Public Key Cryptography Authentication: A type of authentication performed using asymmetric cryptography algorithms. The most popular asymmetric cryptography algorithms are Rivest–Shamir–Adleman (RSA) and elliptic-curve cryptography (ECC). These algorithms are also used in the design of secure sockets layer / transport layer security (SSL / TLS) (Thomas & Wiley, 2000) and secure electronic transaction (SET) (Bella et al., 2003) network security protocols used for the security and data privacy of the systems. In this structure, the authentication process is realized by using the private key in the length and range limited by the method based on the asymmetric cryptography algorithm used. The public key is derived from passing the private key through specific mathematical calculations that vary according to the cryptography algorithm used. It is not possible to acquire a private key using the public key. The generated public key is used as an identity and can be freely shared over the Internet

Single-Sign-On Authentication: In this method, a central login system is used as a means to authenticate at the intended computer system. This authentication method has been created to prevent separate passwords and remembering for each site and also to eliminate the need for trust against the target site. After logging on to a central system, authentication is not performed again when attempting to connect to a website that accepts this system tokens (Babaeizadeh et al., 2015; Peyrott, 2015). Centralized systems form the single-point-of-failure point of the designed structure. Thus, when a topology is created with this authentication method, the authentication system becomes the single-point-of-failure point of the entire topology (Sirota, 2017). At the same time, the entire authentication process is managed by a 3rd party foreign source, which raises serious security concerns.

Out-of-Band Authentication: This authentication method is realized by sending information such as a token, user name, and password required for the authentication process over another communication source except for the primary communication source that is communicated during the authentication process. The purpose of this application is to protect the account from malicious persons in case the privacy of the primary communication channel has been violated by malicious persons. The security

of this method is ensured by the use of two separate communication channels which are not directly related or connected to each other. It is unlikely seen that these two channels are under the control of the same malicious person (Wu et al., 2018). Applications such as push notifications, short message service (SMS) and phone calls sent to mobile phones using cellular networks instead of the internet are the most commonly used examples of out-of-band authentication (Pandya et al., 2015; Rouse, 2014).

1.3.3 Authentication Schemes

Single Factor Authentication: It is the most simple and basic authentication scheme. In this scheme, only one authentication factor is used in the authentication process. Due to its user-friendliness, it is still the most commonly used authentication scheme today. Due to the most commonly used authentication method is password authentication in this scheme, this scheme is considered to be a weak scheme (Ometov et al., 2018). However, password authentication is not the only authentication method used in this scheme. It is possible to perform a robust authentication process by using authentication methods, such as biometric authentication, where it is not possible to have a security breach due to user factors (Khan & Zhang, 2007; Rouse, 2015). Scheme of work is illustrated in Figure 1.3

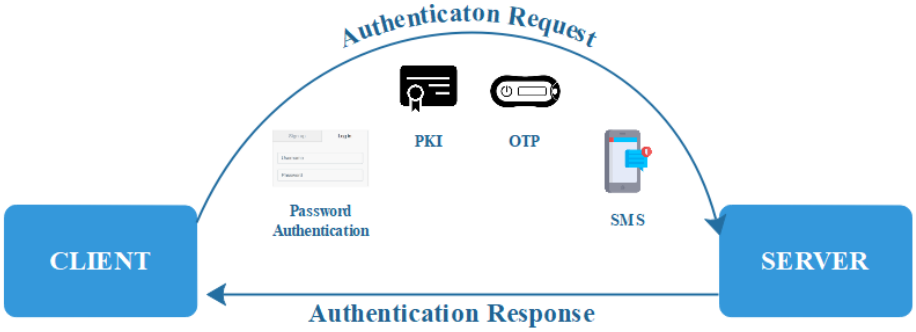

Figure 1.3 Single factor authentication

Multi-Factor Authentication: In this scheme, two or more authentication factors are used together to provide a higher level of security on information systems. In this scheme, “what you know” and “what you have” factors are generally used together

(Scheidt & Domangue, 2005). Thus, an additional layer of security is created as a precaution against attackers. The most common usage that we face today is online banking systems. When you want to log on to a banking account, the one time password (OTP) code is sent to the mobile phone of yours by the bank (Aloul et al., 2009). A simple workflow is illustrated in Figure 1.4

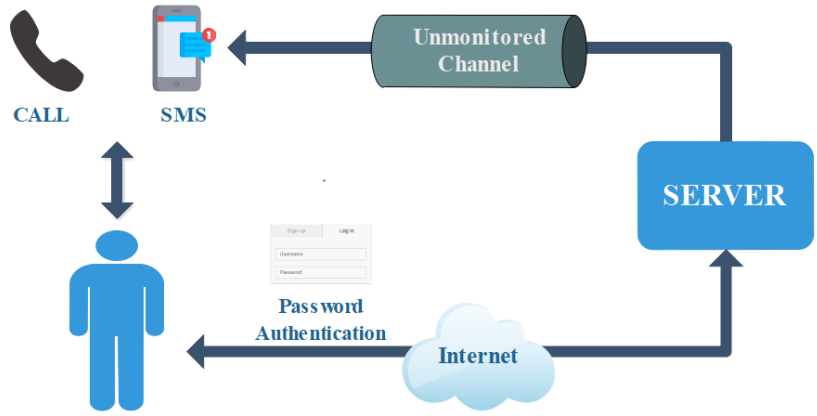

Figure 1.4 Multi-factor authentication

Mutual Authentication: Although the authentication process is generally seen as the authentication process on the target to be connected, the communicating parties should not have doubts about each other's identities. In this scheme, public-key cryptography is mostly used, but it is not a must. For example; after logging in to the bank's website, before the SMS verification step, the bank's web page displays the image previously specified by the user, thus ensuring that the user is logged in to the actual bank site (Spacey, 2016). The TLS cryptography protocol used in the Network Layer is the most common protocol that provides mutual authentication. The purpose of this protocol is not only to verify the identity of the parties in communication but also to determine the secret key to the symmetric cryptography algorithm that will be used to ensure communication confidentiality. Confidentiality, authentication, and integrity can be achieved in the communication between the parties by using the TLS protocol. Simple TLS handshake process is illustrated in Figure 1.5

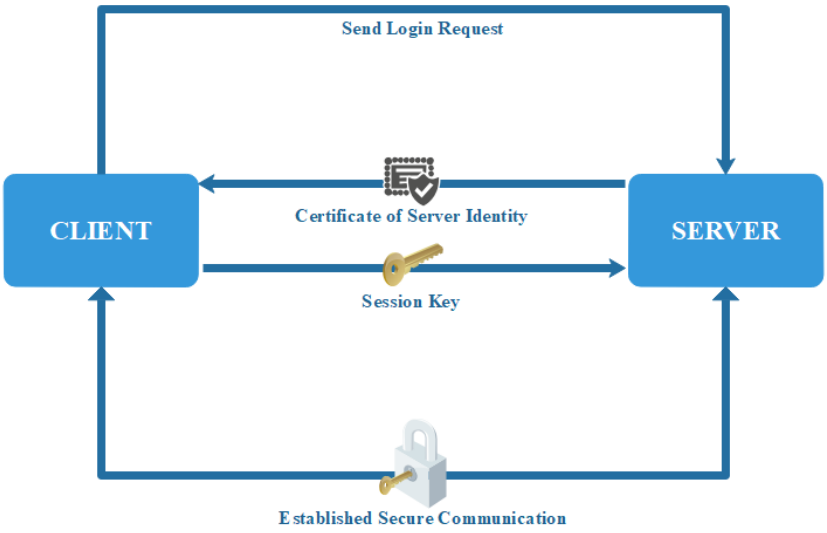

Figure 1.5 Simple TLS handshake

1.3.4 Problems on Authentication Systems

1.3.4.1 Single-Point-of-Failure Problem

As a general definition, single-point-of-failure points of a system or circuit are defects made during the design, implementation, or configuration process of that circuit or system. These defects are risk points that may cause the system not to work entirely and even worse, to crash completely (Rouse, 2012). Single-point-of-failure is a more comprehensive definition of single-point-of-compromise. Single-point-of-failure points are possible at every point in a system design process. These points, especially if they persist on a critical process like authentication, are the risk points that may cause the designed system to stop or crash altogether. Since these errors are caused by developers in the design process, standardizations are needed to find a permanent solution to this problem. Network connection and electrical power outages in information systems are seen as the most critical single-point-of-failure points that affect the business negatively and cause money loss (Bressler & Bergen, 2014). But in the event of a failure in authentication systems that are responsible for maintaining data accuracy and data integrity, it can cause the elimination of confidentiality and also guaranteeing the integrity of the data stored anywhere within the system may be no longer possible.

"What you are" factors are single-point-of-failure points when used alone. For people; palm, iris, and fingerprints are biometric data that identify their identities. However, these pieces of information can be seized by malicious applications or tampered readers. Thus, the use of biometric data on an authentication process, which is seen as a compelling identity verification method, becomes a critical security breach since biometric data cannot be changed once seized (Michaluk et al., 2013).

Centralized authentication management systems within traditional or IoT networks also form a single-point-of-failure point. These points, where all credentials and keys are stored and easy to access, are the main target of attackers who want to infiltrate this network (Hughes, 2015).

Today, the popularity of single-sign-on systems is increasing. There are many geographically dispersed single-sign-on products on the market. These systems are being developed with considerable investments, and therefore, the developers entrust the authentication processes of the products they developed to these authentication systems. However, a security vulnerability has been identified on the Security Assertion Markup Language, which was recently used in the implementation of Single-sign-on systems, allowing malicious hackers to log into the system by impersonating authorized user's identity (Whittaker, 2018). OneLogin, one of the infrastructure providers for single-sign-on authentication, had been the victim of attackers, they reported that encryption keys that protect users' passwords were in the hands of the attackers (Hoyos, 2017).

1.3.4.2 Trusted Third Party Problem

We use banks in transactions such as money transfer and the purchase and sale of precious goods (Szabo, 2001). Similar to that, in the majority of authentication schemes, trusted third parties are used to satisfy the other party's identity. These schemes require to work that each node in communication blindly trusts the trusted third party node, which means that parties blindly accept the trusted third party and the communication channel is entirely secure and free from intruders. Developers and

users have limited authority over trusted third parties. When using a blindly trusted channel, operations such as monitoring and taking measures are not possible by designed system supervisors (Levi & Caglayan, 1997).

As an example, in single-sign-on systems, users store their passwords on SSO servers, and systems need to rely on the authentication tokens that will be sent over the SSO system. The systems have no control over SSO, and there is no choice but to accept the credentials sent by the SSO correctly.

The most commonly used channel for out-of-band authentication is the cellular network. Two-factor authentication is performed with OTP codes sent over the mobile network. The communicating parties do not have any authority over the mobile network used in this embodiment. This use was considered unsafe by the National Institute of Standards and Technology (NIST) and recommended not to be used in the draft publication issued in 2016. However, in the final release, this article was updated with that OTP token can be sent to a predefined device or interconnected telephone line (Unen, 2017). It is proven that Signal System 7 (SS7) protocol, which is used for routing and SMS routing in the global network, is not sufficiently secure. The attackers took advantage of the weaknesses of this protocol and redirected the OTP codes sent by the systems to take control of the numerous accounts. As a result, lots of bank accounts were emptied (Gilsenan, 2018).

There is also a trusted third party problem in the TLS algorithm, which is often preferred for Mutual Authentication. In this algorithm, a certificate authority is used as a trusted third party to confirm identities in the authentication phase. In this protocol, it is blindly accepted that the certificate authority will never be the perpetrator and cannot be taken over by anyone. In public-key based systems, there is no need for continuous connection with the certificate authority, proving a certificate that signed by trusted authority is sufficient to confirm claimed identity (Levi & Caglayan, 1997). However, when the private key of this certificate authority is taken over, it is possible to generate certificates on behalf of everyone and attackers can get authenticated on systems that trust this authority; even worse, this authentication mechanism does not

provide any revoke options in its design. This problem has led to the need to continuously check the revoked status of the certificate over another layer by the node that wants to authenticate the other party over a network connection. As a matter of fact, many certificate authorities like Comodo, Verisign have been victims of attackers to date (Latif, 2011; Menn, 2012). Moreover, Verisign was removed from the trusted authority list of browsers for producing false certificates (Digicert, 2017).

1.4 Introduction To Blockchain

We require a trusted third party for asset management and money transfers in our daily lives. We use banks to meet this requirement. Blockchain technology, which was developed to eliminate this requirement in money-based transactions, became known in 2008 with an academic article (S, 2009). Blockchain is, in fact, a perfected version of the combination of works developed to date. The non-changeability of data, the biggest gain thought to be brought by the blockchain, was initially been proposed on a study aimed at undoubtedly confirming that the documents were not tampered or damaged by signing and timestamping documents (Stornetta & Haber, 1991). The idea of storing these verification data with the Merkle tree was taken from the improved version of the same study (Bayer et al., 1993).

In the Blockchain network, each node carries all transactions, and the accuracy of these transactions is ensured by using the Merkle tree structure. By using the Blockchain network, Ownership of an asset can be verified without using a trusted third. Public key cryptography is used to supply ownership information within this network.

Blockchain consists of blocks attached together. A block is created at changing time intervals, and each new block represents the current status of the blockchain network (Shorish, 2018). Therefore, blockchain is also referred to as an infinite state machine. The consensus is required for every new block created within this network. There are several consensus methods developed for blockchain to achieve consensus.

Blockchain networks that are categorized into two main groups which are namely, public blockchain network and private blockchain network, are explained below

- **Public Blockchain Network:** Anyone can join this network and download all of the Blockchain network data. All data in the Blockchain network is synchronized with P2P protocol within all nodes that members in the network. Therefore, the consensus algorithms must be capable of verifying and proving each data that is written on the Blockchain network over each node that members of the network.

- **Private Blockchain Network:** In this type of network, the blocks published for inclusion on the network are only approved by a central or distributed located authorities. When a private blockchain network is used, it is possible to specify the nodes that will join the network. Thus this feature ensures the data within the network to be received only by those authorized nodes. In this study, a private blockchain network implementation is preferred, and only specific nodes have been approved to be a member of this network.

The first cryptocurrency based on the Blockchain technology was introduced in 2009 under the name Bitcoin. Although Bitcoin is the first cryptocurrency to use this technology, many cryptocurrencies using blockchain have emerged and continue to appear. Based on the idea of not limiting Blockchain technology to cryptocurrencies, a white paper, defines and describes Ethereum, was announced in 2013 (Wood, 2017). The most significant innovation with Ethereum is that it was now possible to perform Turing-complete programming over the blockchain network with the Ethereum Virtual Machine environment (Sellin, 2017). Ethereum is an open-source, distributed computing platform with Blockchain-based intelligent contract functionality. Ethereum enables application developers to develop and deploy decentralized applications.

1.5 Motivation

The requirements for confidentiality, availability, and integrity that already exist for the traditional Internet network have reached a more critical dimension with the increasing popularity of the Internet of things technology. In addition to the increasing number of devices, a significant increase in the rate of confidential data obtained from the devices caused this situation. Moreover, it is evident that these requirements can not be fully met with traditional approaches and methods. Our motivation in this study is to secure the Internet of things environment by providing an authentication&authorization control, and confidential communication mechanisms, which are based on recently emerged Blockchain technology that eliminates trusted third party requirement. These mechanisms are the most critical rings in the chain of security that satisfy these requirements.

1.6 Contribution

The main objective of the thesis is to provide a robust security mechanism that provides confidentiality, availability, and integrity for the Internet of things, whose usage area is increasing each day even many security concerns remain unresolved. With well-known technologies and techniques, it does not seem possible to provide a security mechanism that can address most of the security concerns. Thus, Blockchain technology, a recently introduced technology, is used to cover the majority of these security concerns in the thesis.

The most striking point in this study is to prove all required operations to communicate over the Blockchain network are also possible for constrained-devices, which is thought to be impossible due to these operations too complex to run on constrained-devices.

The secondary objective of this study is to strengthen the security of the application layer protocol which cannot provide an authentication mechanism other than a simple

username and password authentication and TLS protocol with the Blockchain intermediated authentication model and payload encryption.

Another purpose of this study is to provide robust privacy with pseudonymous encryption on the Blockchain Network.Thus, the confidentiality concerns over the Blockchain network are also addressed in the thesis.

As a final contribution, this study is analyzed against security concerns on the Internet of things networks by using the application developed for this study, and also some performance measurements on ESP32 MCU, which is picked to use as constrained-device in this study, is given.

1.7 Outline of the Thesis

The problems experienced with IoT systems and various studies conducted before this thesis are examined in the second chapter. In the third chapter, there are necessary explanations of the technologies used during the design of the proposed mechanism.

In the fourth section, the hardware and software equipment used are explained, and the method proposed step by step is explained in detail.

In the fifth part, the security analysis of the proposed method is presented, and the data and graphics showing the performance of the device on the blockchain network are presented.

In the sixth and final section, a general evaluation has been made about the study, and future improvements have been mentioned.

CHAPTER TWO
RELATED WORKS

The security violations on the IoT networks and studies covering security requirements, potential security violations, and authentication methods were examined in detail. Through the primary concern of our work is constrained-devices, constrained-devices, and these capabilities were also another literature review topics in this thesis.

2.1 Literature Review

Harsha M. S. et al. (Harsha et al., 2018) identified publicly accessible MQTT brokers and observed that 60% of the brokers they selected as samples allowed access to all topics without authentication and authorization.

In 2016, a distributed denial of service (DDoS) attack was launched against a DNS service provider, which caused Netflix, Twitter, and GitHub to be out of service for long hours. According to analysts, it was found out that attackers took over 400.000 devices by taking advantage of weak credentials (Kolias et al., 2017).

To the best of our knowledge, there is no blockchain authorization study directly similar to this study, which includes authentication and pseudonymous communication of *things* over a blockchain network. The remarkable point of this study is that a unique secret key is created anonymously inside two endpoints via blockchain and that key is used for encrypted communication between two endpoints over both the blockchain and traditional network.

Yerlikaya & Dalkilic (2018) proposed a mutual authentication and authorization model for the MQTT application layer protocol. This model uses the OAuth2.0 protocol. The Access Control List used for authorization is stored on the WSO2 server. For the mutual authentication process, MongoDB is used to store key and counter values used in the HMAC-based one-time password (HOTP) algorithm and HOTP

algorithm. The HOTP algorithm is applied five times in order to increase hardness, and the results are stored in MongoDB per authentication process.

Amrutiya et al. (2019) proposed a two-way authentication framework using the blockchain network and smart contract. They have developed authentication methods with the Pluggable Authentication Module (PAM). Connection to Ethereum Blockchain is established via the javascript object notation remote procedure call (JSON RPC) application programming interface (API) interface provided by the Ethereum Client node. Three different user groups are created on the smart contract, including users, managers, and servers. Unique Ethereum Address is produced for each device and user. In this structure, the user can read all the servers defined in the system from the smart contract. It sends a request for the server to which it wants to connect via a smart contract, and this request is approved or rejected by the authorized person. After the connection request is confirmed, the user generates a 6-digit numeric OTP code and hashes it with the SHA3 algorithm and stores it on the smart contract. There is no clear definition of how the user stores the data on the smart contract after the request has been approved by the authority. Since the blockchain is an open ledger that can be seen by everyone, the generated code will be easily gathered by attackers. They did not propose a salt mechanism on the hashing stage. A malicious user can crack a SHA3 hash output of 6 digit numeric input in a minute. Even if the salt mechanism is used, the OTP codes on the entire system can be broken after a node capturing attack. At the same time, a real blockchain network has not been established; instead, the ganache test environment is preferred. Therefore, transaction validation and block creation times have not been evaluated, and after sending the user's OTP data to the Smart Contract, the time required for transaction validation is not included. It is unclear when the user can continue to proceed to the normal login process. Finally, this system, which is proposed as Trustless authentication, connects to the blockchain network via Ethereum JSON-RPC API, which operates externally on proposed work. Hence a mechanism for verifying the data received from this connection is needed, which is not proposed. Thus, the requirements for the trustless environment have not been met.

Auth0 is the authentication and authorization management platform. This platform can be used in Web, Mobile platform, and legacy applications. The authentication and authentication processes for protocols used in the IoT application layer, such as MQTT, can also be performed with this platform (Auth0, 2018). The main target of this platform, which works with the OAuth2.0 protocol, is to implement a Single-Sign-On process to log on to multiple applications with a single credential. This platform supports a variety of authentication methods. One of these methods is authentication with using the Ethereum Blockchain Platform (Sebastian Peyrott, 2017). In this work, the input request is sent over the traditional network using the Ethereum address, the node that receives the request creates JWT token then signs. After the signing process, the node sends created JWT token to the requester. In order to complete the login process, this JWT token must be sent to the predetermined smart contract address. However, the Ethereum address of the smart contract responsible for the login process and the login method to be used must be shared with the user in advance. Besides, Auth0 has a centralized structure, and the authentication process is managed by central servers. Therefore, the advantages of the distributed and decentralized structure, which is the crucial feature of Blockchain, cannot be used with this platform and also this platform will constitute the single-point-of-failure point of the system to be designed.

Randa Almadhoun et al. (2018) proposed an authentication scheme for *things* using blockchain. In the proposed architecture system, the Ethereum address is created for each *thing*, gateway, and user. However, this design only includes two-factor authentication between the user and the gateway, and the *things* are not located on the blockchain. In the proposed scheme, after the authentication process, authentication between the user and the *thing* is not considered. Instead, a direct SSL connection is proposed. It is a requirement that *thing* needs to validate the user's identity and authority, which is not mentioned in this work.

Siye Wang et al. (2018) brought forward an HTOP-like mutual authentication protocol for RFID tags using blockchain. In their proposed structure, each RFID tag is defined with Ethereum addresses, Ethereum Balance is used as the moving-factor value during the mutual authentication process, and after each successful transaction,

the determined amount of money is sent to the address, and the new moving-factor value is formed. However, blockchain is a public distributed ledger and account balances are publicly visible. In addition, how the Ethereum address is generated for RFID tags is not clearly defined.

Oscar Novo (2018) put forth scalable access control architecture. Each *thing* has an Ethereum address, and the entire access control list is stored in the blockchain network. In this study, things use a constrained application protocol (CoAP) as an application layer protocol, datagram transport layer security (DTLS) as a transport layer security protocol. T*hings* are connected to the blockchain through the management hub, *things* do not send or receive requests to the blockchain network. Additionally, there is no information about how the *thing* and the hub authenticate each other. At the same time, the management hub does not validate the response it receives from the miner node requested for authentication.

Suarez-Albela et al. (2018) implemented the RSA and ECC algorithms using the ESP32 device and performed performance comparisons using different curves and parameters with these algorithms. Among the curves used during the performance comparison during their studies, the secp256r1 curve, which has the same complexity as the secp256k1 curve used in the Ethereum network, was used. Energy consumption measurements were also performed for this device with limited resources.

CHAPTER THREE
PROVIDING TRUSTLESS AUTHENTICATION AND CONFIDENTIALITY MECHANISM FOR INTERNET OF THINGS

Blockchain technology can be a powerful solution for scalability and authentication issues which are considered as the most severe problems in the Internet of things networks without the need for a trusted third party. In this section, details of technologies behind the proposed mechanism provides secure authentication and confidentiality to *things* on the Internet of things networks are given.

3.1 Ethereum Blockchain

Ethereum Blockchain is a ledger technology that allows people to write and run programs and develop their cryptocurrencies over the blockchain. The default cryptocurrency name used in this structure is Ether. The logics and mechanisms used for verifying transactions, for ensuring that these transactions cannot be changed, and for supplying the latest status of the blockchain, is similar to its successor, Bitcoin. However, The Ethereum blockchain stores the transaction list and the most up-to-date status of the structure together (Buterin, 2013). Ethereum also has shorter block creation time with the aid of a modified version of the inclusive blockchain protocol it uses, mainly because to achieve low creation times (Lewenberg et al., 2015).

In order to initiate a request, an account on the preferred blockchain network is required. There are two different types of accounts. One of them is called an external account, and the other is called a contract account. Ownership of an external account is proved by public & private key pair, and all transactions and messages within this network can only be created by these accounts.

3.1.1 Genesis

The Genesis block is a very first block in the blockchain. It represents the starting point for the blockchain network. Since it is the first block, it does not contain the address of the previous block. The properties of the blockchain network, the balances of the initial accounts are defined in the Genesis block. These definitions load the config files to create the Blockchain network with the Genesis block containing initial information.

The Genesis block is not transferred peer to peer by connected nodes. Therefore, the genesis block is needed to be predefined to the nodes wants to connect the blockchain network individually. The transfer is only possible from the first genesis block (Arvanaghi, 2018). Genesis block inside chains in the Blockchain network is illustrated in Figure 3.1

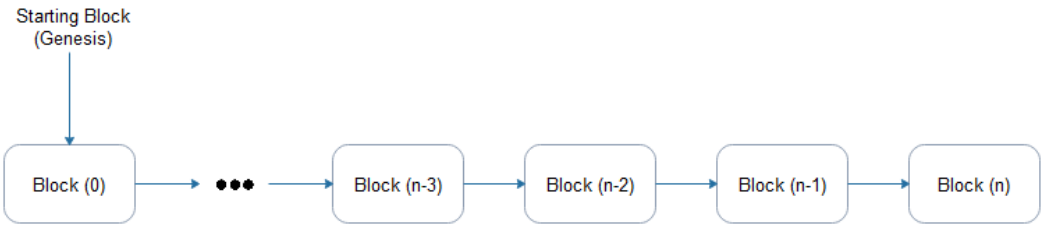

Figure 3.1 Genesis block on the Blockchain

3.1.2 Merkle Proof

The Merkle proof structure is the logic used to check the accuracy and integrity of the data contained within the blockchain network. Merkle proof is used as a solution for the following factors in the Blockchain network (Prahalad, 2018).

- To see if the data belongs to the blockchain network.
- To verify the validity of data that is part of a data set without storing the entire data set.

- To verify the validity of a given data set, prove its existence in a large data set, without revealing the block data set or a subset of that set.

Merkle Proof is consists of using the binary Merkle tree structure. In the most general sense, a Merkle tree is a way of gathering a large number of pieces of data. A piece of data is created with one-way hashing methods (Buterin, 2015). The popularity of one way hashing methods has increased after they prove their benefits on the Merkle tree, which has found use on the blockchain network. The following three types of the Merkle tree is stored within an Ethereum block.

- State Tree
- Transactions Tree
- Receipts Tree

Thanks to this structure of Ethereum, it is possible to check the validity and integrity of data without downloading all transactions within the entire network. Thus, non-miner nodes can be able to ensure the accuracy and completeness of the records within the blockchain such as transactions and events by downloading only block information which is significantly smaller than transaction records (Kasireddy, 2017).

When the Transaction Tree in a block containing four transactions is examined as an example, a binary Merkle tree with three-tier is shown in Figure 3.2. Each leaf node of this tree includes a hash of a data set. The hash in a parent node is obtained by combining and re-hashing the hash contained in the two children node. This process is repeated until only one node without a parent, remains. The only one node is named as the root node is present in the top layer.

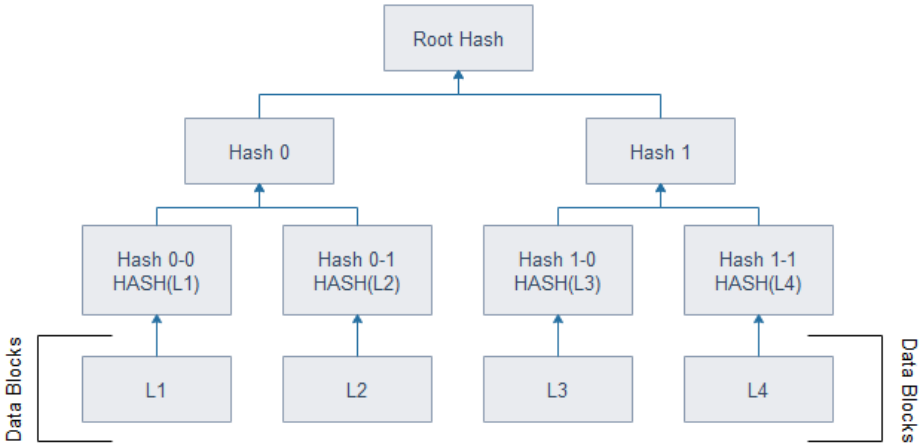

Figure 3.2 A simplified Merkle tree

Merkle proof takes advantage of the way Merkle Tree works upwards. Thus, A change to leaf nodes will cause to change the hash information in the transaction root node. This system, which uses the hash mechanism, does not require the use of real data.

3.1.3 Distributed Consensus

The Distributed consensus protocol is the core of blockchain and forms the heart of blockchain technology. The need for Trusted Third Party is eliminated through consensus protocols. Blockchain uses this protocol to negotiate the data to be added to the public ledger. The most important detail here is that no node or authority decides which data to add to the system. Instead, data to be added to the system is approved by the majority of nodes within the network.

The fault-tolerance feature on the Blockchain network is achieved by the redundancy and decentralized governance qualities of the consensus algorithm used. In the blockchain network, immutability for the data can be guaranteed as long as the majority of the nodes are honest. This situation on computer networks is called The Byzantine Generals Problem (Lamport et al., 1982).

Proof of Work and Proof of Authority, the consensus algorithms used by the Ethereum blockchain network, are described below.

3.1.3.1 Proof of Work

The first proof-of-work like algorithm was introduced in 1999 (Castro & Liskov, 2002). The aim of this algorithm was to find a solution to the Byzantine General Problem (Seibold & Samman, 2016). In summary, byzantine general problem; In a city that was besieged during the war, generals scattered all over the city could agree on a war plan between each other. Proof of Work algorithm is presented as a solution to a similar problem for distributed systems.

Proof-Of-Work is the consensus algorithm proposed with bitcoin in 2008 and works with the logic of solving an algorithmic problem and obtaining a numerical value. This process is like solving a puzzle. After a node acquires a numeric value from calculations, it shares that value to all nodes in the network. Other nodes in the network can verify the accuracy of this value by doing mathematical operations. The longest chain ever produced in this network is considered correct. As long as malicious nodes do not have the majority of processing power on the network, with this feature, the accuracy and integrity of the data are guaranteed without the use of a trusted third party.

In order to increase the number of honest nodes within the Public Ethereum Blockchain structure, there are mechanisms such as rewards varying overtime per block created and paying miners per verified transaction. However, the difficulty of the puzzle needs to be solved to create a block and get a reward, increases with time, and this will reduce the number of honest nodes within the network. Blockchain networks use PoW will become more vulnerable in the future.

Another problem with this consensus algorithm is its excessive energy consumption. Some research revealed that the total electricity consumption of the Blockchain networks exceeded the total electricity consumption of many world countries (CIA, 2016; Narayanan et al., 2015; Vincent, 2019; Zhang & Wen, 2017).

In private blockchain networks that are set up for use in the Internet of things networks, the use of PoW is unsuitable for the reasons as mentioned above. Firstly, the number of nodes is minimal in private blockchain networks, and raiders can quickly get required the majority of the processing power. Secondly; energy consumption and the use of processing power are critical in devices with constrained resources.

3.1.3.2 Proof of Authority

In the Proof of Authority consensus algorithm, complex problems are not be solved to validate transactions, but instead, there are predefined validators in this system. These pre-selected validators are responsible for creating blocks within the blockchain network and authorizing transactions. A successful attack on the Ethereum test network revealed the need for such a consensus algorithm (Hertig, 2017). Since there is no mathematical puzzle that needs to be solved in order to verify transactions, processing power is hardly consumed.

The Proof of Authority model works with a partial trust mechanism for specific credentials within the network. Therefore, in order for a node to become an authenticator in the network, the identity of the node must be proven and verifiable. This model also includes a trust system. The authenticator uses its reputation for a transaction or a block it creates. The probability that a validating node and issue a new block are directly proportional to its reputation on the network. The likelihood of a low-respected node to create a new block is equally low. Therefore, the validating nodes try to protect their reputation within the system (Yaga et al., 2018).

Similar to the PoW consensus model, the presence of N / 2 + 1 honest nodes in this model is sufficient for the safety of the blockchain network. Not all signer nodes need to be always online. In this way, the fault tolerance of the blockchain network is persisted. At the same time, a new authenticator node can be added to the blockchain by a majority vote of current authenticator nodes, besides a malicious node in the network can be expelled from the system in the same way (Nolan, 2018).

In this miner-free structure, block and transaction validation time is not depend on processing power; instead, the time is fixed at the time of blockchain creation. Thus, it is possible to obtain lower block creation times with the Proof of Authority consensus algorithm, which can be obtained with very high processing power or even it is not possible to obtain by using other consensus algorithms like Proof of Work.

3.2 Ethereum Virtual Machine

Ethereum Virtual machine is the name of the structure that makes programming within the Ethereum Blockchain possible. Each Ethereum Blockchain client node contains the Ethereum Virtual Machine. Ethereum Virtual Machine's primary responsibility is to provide an isolated environment from the network where the node is located or from all systems belonging to the node to run smart contract bytecodes (Bitrates, 2018).

Like the Java Virtual Machine (Lindholm, Bracha, Buckley, & Yellin, 2013), the Ethereum Virtual Machine has its own opcodes. Its 140 unique opcodes all make the Etherem Virtual Machine Turing-complete. A Turing-complete virtual machine means that complex programs can be run within this environment, provided system resources are sufficient, and enough payment is made (Hollander, 2019).

There are many programming languages available for developers to run in the Ethereum Virtual Machine, which generates EVM opcode output when compiled. Although many programming languages such as Serpent, Vyper, LLL, Muıtan are currently available, the Solidity programming language is the mainly used programming language used in the Ethereum network (Orlicki, 2018).

3.2.1 Accounts

Only Ethereum accounts can perform the Ethereum network state change. An Ethereum account object contains 20 bytes of address, the number of transactions (nonce), storage state, instinct balance, EVM code, and all accounts are maintained as

part of Ethereum state (Tudonu, 2018). In the Ethereum Blockchain network, there are two types of accounts, which are identical to each other, described below.

Externally Owned Account: This account is managed by a person or device with the public & private key duo that is appropriate to the Ethereum network. Address information is generated from the public key with the Keccak-256 hashing algorithm (Badretdinov, 2018). This type of account can transfer money by creating transactions on the network or trigger EVM code by calling a smart contract function (Kenneth, 2018).

Contract Account: This account is controlled and owned by the EVM. The address of the contract account is derived using the EOA account that created them and the current nonce value of that account at the creation time. It contains an EVM code and the storage state which is used by the EVM code. Code execution operations are triggered by the EOA and can perform Turing-Complete operations after triggering (Kenneth, 2018).

3.2.2 Gas

Ethereum Gas is used as a cost unit in the Ethereum Blockchain. Any transaction is made in the blockchain network causes a gas cost, which is deducted from the account that started the transaction and paid to the account that performs the validation. Some costs are defined as fixed in the Ethereum Virtual Machine varying to the process (Consensys, 2016). According to the increasing complexity of the actions performed within the smart contract, the cost increases, and more costs must be paid by the requesting node to complete these actions.

Gas also has a protective role against flood attacks that can be done on the blockchain network by malicious users. Ether balances in the attacker's accounts will expire after some time, regardless of whether it is a public or private blockchain because each transaction sent on the blockchain network cause an additional cost. Thus, the attackers will be prevented from permanently damaging the blockchain

network. For example, in the current state of the Ethereum public network, it is necessary to pay millions of dollars to perform a simple attack (Hertig, 2017).

3.2.3 Transaction

State changes on the blockchain network are performed using transactions. There is one transaction structure, but using this transaction structure, the following three different operations can be performed on the blockchain network.

- Money transfer between two accounts
- Function call on a Smart Contract
- Deploy a smart contract into the blockchain network

The following fields are included in a transaction structure (Pegasys, 2018).

- to: If the transaction is a smart contract call, this field contains the address of the called smart contract. If the transaction call is a money transfer, then this field contains the destination Ethereum address. When the purpose is to create a smart contract, this field is left blank.
- nonce: This is the area used to avoid the double-spent problem. Technically, the number of transactions sent over an account. For each sent transaction, the value here is incremented by one. When a transaction with the same nonce value is sent, it will not be accepted into the blockchain network.
- gas-price: For the gas cost incurred during transaction executing, it is the value that includes the fee payable per one gas unit. Miners will first approve transactions with a high gas-price value to increase their earnings, and transaction approval times will be reduced in the blockchain networks using the PoW consensus algorithm.

- gas-limit: This is the field in the transaction that determines the maximum cost accepted by the sender. Especially in Smart contracts, this area becomes essential because the cost is not absolute.

- value: The amount of cryptocurrency to be transferred is entered in this field.

- v,r,s: When a transaction is signed with Elliptic Curve Cryptography, the output of signing operation is written to this field.

- data: This field is used to send messages via transaction. If the operation is a smart contract call, the parameters to be used during the call are added to this field. Some clients virtually reference this field as the input field.

- init: This field is used only during the smart contract creation process, and the compiled EVM code output of the written program is placed in this field.

The most significant point here is that there is no "from" property in the transaction structure. "from" property is obtained and showed virtually by using v, r, s objects, which are the signing algorithm outputs in the transaction structure (Antonopoulos & Wood, 2018).

3.2.4 Smart Contract

Fault-tolerant and safe distributed application built in the Ethereum blockchain network is called a smart contract. Once the smart contracts are created, the methods and conditions contained in it cannot be changed, and the data stored in it can not be changed. It is referred to as a smart contract because of its ability to trigger certain events if predefined conditions are provided (Musienko, 2019).

Smart contracts use the blockchain network as storage, and the maximum data size that can be stored in these contracts is theoretically unlimited. But; a gas fee is required each time a storage operation is performed (Senta, 2018). Two types of smart contract execution are detailed below (Hitchens, 2018).

- Contract Call: All blockchain nodes store smart contracts and their data. Therefore, it is possible to obtain read-only smart contract data over the connected Ethereum node. There is no cost during read-only data acquisition. It is not a real smart contract operation and does not appear in the transaction log.

- Contract Transaction: It causes the smart contract code to be executed by all nodes connected to the blockchain network. Therefore, the cost in gas units occurs. This type of call may change data within the smart contract storage. Moreover, the smart contracts can send events on the blockchain network. The cost of the transactions performed by the smart contract is paid by the sending account.

With these features, smart contracts make it possible to develop fault-tolerant, redundant, and robust applications.

3.3 Keccak Hashing Algorithm

Algorithms that produce a fixed-length output using mathematical functions for the input of any length are called hashing algorithms (York, 2018). Hashing algorithms are the most common examples of one-way algorithms. It is not possible to recover the original data using the output generated by these algorithms. These algorithms are used to create a digital fingerprint for the data at hand. By using this fingerprint, data integrity verification or data identification operations can be performed (Roussev, 2009).

The most common hashing algorithm used in the Ethereum Blockchain network is the Keccak-256 algorithm. All Merkle trees used in the Ethereum are generated from

the outputs of this algorithm (Kim, 2018). This algorithm is also used in the digital signing process of a transaction. Similar to other digital signing processes, transactions in Ethereum are first summarized with the Keccak-256 algorithm. Signing is performed using this hashing output (Rouse, 2019).

It is safe to sign a digital fingerprint created with a robust algorithm such as Keccak-256 because the smallest change to the original data will completely alter the algorithm output. Besides, the Keccak-256 algorithm is resistant to attacks and has proven its reliability (Dinur, 2015).

The Ethereum address is also derived from the public key using the Keccak-256 algorithm. The Ethereum address is the last 20 bytes of the hash output of 32 bytes (Witherspoon, 2017).

In this thesis, the Keccak-256 algorithm is used in the production of tokens in the smart contract, during communication on the Ethereum blockchain and on the rest of the authentication process on the traditional network.

3.4 RLP Encoding / Decoding

Object serialization is a process applied when a data structure needs to be transferred over a network. RLP Encoding is the serialization format used in the Ethereum Blockchain network for this purpose. After the transaction structure is created in the Ethereum environment, object serialization is performed, all transactions in the Blockchain network is in RLP-encoded format. (Antonopoulos & Wood, 2018).

3.5 Elliptic Curve Cryptography

Elliptic curve cryptography is a public-key cryptography method developed based on a discrete logarithm problem. Signing on the Ethereum blockchain network is performed by this method. In this study, this cryptography was used in the proof of identity stages. The most significant benefit of this cryptographic algorithm is that it

can provide the same level of protection with a short key length compared to its competitors. Performance research has shown that elliptic curve cryptography is the optimal choice for constrained devices (Suarez-Albela et al., 2018; Suárez-Albela et al., 2018).

Ethereum Blockchain makes use of a specific curve and a group of mathematical constants when using elliptic curve cryptography. The name of the standard in which these curves and mathematical constants are defined is secp256k1 (Antonopoulos & Wood, 2018). The curve $y2 = x3 + ax + b$ ($a = 0$, $b = 7$) of this standard is shown in the Figure 3.3 (Song, 2017).

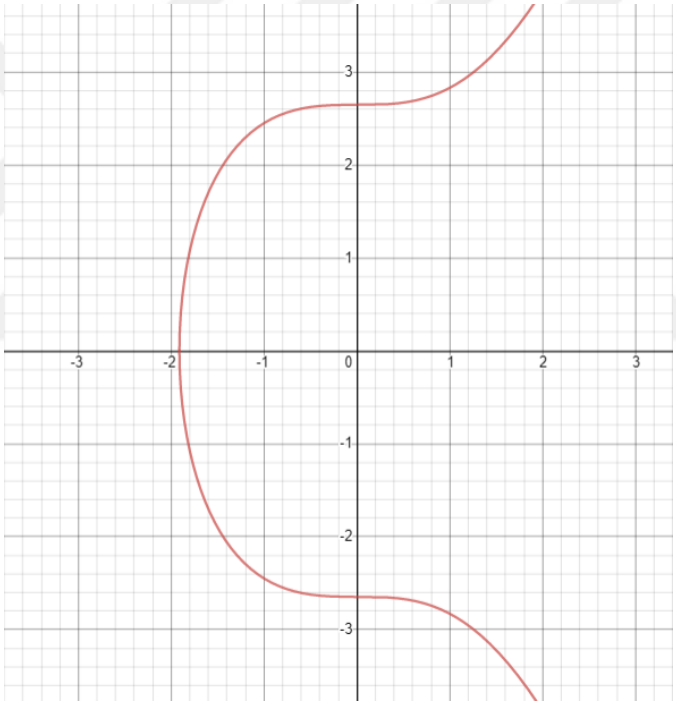

Figure 3.3 Curve belongs to secp256k1 standard

3.5.1 Creating Public & Private Key Pair

A private key is a random number generated in size appropriate to the algorithm used, and the security of the system depends on the randomness complexity of that number. Randomness is ensured by the entropy variety used (Bryk, 2018). The length and range of the random number to be generated are determined by the curve used. The private key in the secp256k1 algorithm used in the Ethereum structure is 2^{256} in size

and has a range of 0x1 - 0xFFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFE BAAE DCE6 AF48 A03B BFD2 5E8C D036 4140 (Song, 2017). This number range is so large that it doesn't possible to reproduce a private key when strong entropies are used.

The public key is obtained by multiplying the private key with the generator point defined in the curve standard used. The result of this multiplication represents a different point within the same curve. This process is considered a one-way process. According to the studies on Elliptic Curve Logirthm Problem, it is not possible to find the private key, which is only used as a multiplier, as long as the private key is produced with strong entropy (Amara & Siad, 2011). As a result, while the public key can be obtained by using the private key, the private key cannot be obtained by using the public key. Therefore, the public key can be shared on the internet with confidence (Antonopoulos & Wood, 2018).

3.5.2 Elliptic Curve Digital Signature Algorithm (ECDSA)

ECDSA is the algorithm used to create a digital signature with ECC public & private key pair in the Ethereum network. A signing algorithm has two objectives. First, it provides solid proof about the message sender. Second, it ensures that the message has not been altered during transmission or storage.

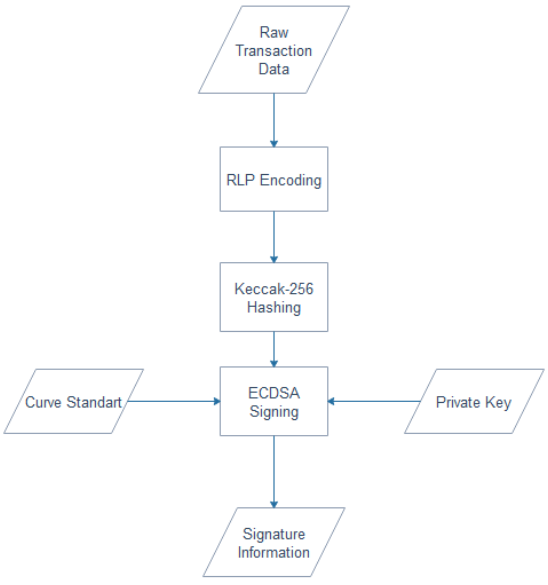

Figure 3.4 ECDSA UML diagram

Users with the private key can perform transaction signing within the Blockchain network. Before signing, the transaction is serialized with the RLP algorithm. Then the hash is generated for the transaction with the Keccak256 algorithm. This hash is signed with the ECDSA algorithm using the private key. The output of the ECDSA algorithm is used as signature information. The ECDSA UML diagram is shown in Figure 3.4 above. The signing process can be formulated as in Equation (3.1) below (Antonopoulos & Wood, 2018).

$$Sig = ECDSA\ (Keccak256\ (RLP\ (m)), k) \tag{3.1}$$

Where;

- m: Created transaction object
- RLP: Algorithm for Serialization
- Keccak256: Hashing function
- ECDSA: Signing Algorithm
- k: Private key of the signer

The output of the ECDSA algorithm consists of a combination of two values. These values are referred to as R and S values. Ethereum standards also use the V value. The purpose of using the V value is to speed up the verification process of the message and to provide replay attack protection by preventing the reuse of a transaction sent within a different blockchain network (Buterin, 2016).

3.5.3 Elliptic Curve Diffie-Helman (ECDH)

In this thesis, the Elliptic Curve Diffie-Helman algorithm is used to create the shared secret key required by the symmetric encryption algorithm to be used in order to establish secure communication between the two parties. With this algorithm, the parties create an anonymous secret key by using the public key of each other without any further communication over any channel.

In order for the ECDH algorithm to work, the parties have to agree on the generator point to be used. The generator point of the secp256k1 standard used in the Ethereum network is also used in this thesis.

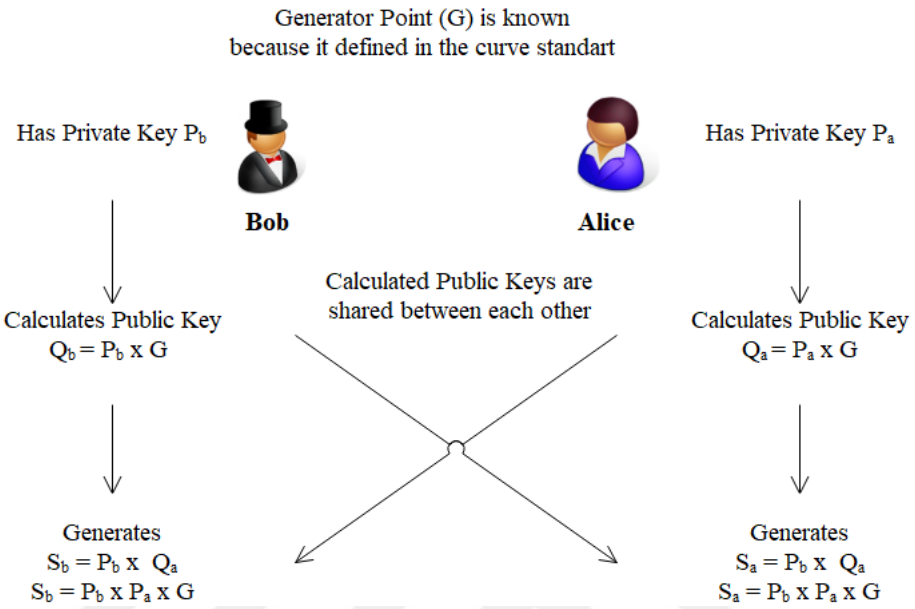

Figure 3.5 ECDH scheme

The way ECDH Algorithm works is illustrated in Figure 3.5, and can be summarized as follows: Let the names of the parties that want to communicate with each other in the encrypted form be Alice and Bob, respectively. (P_a, Q_a) Alice's Public-Private Key pair, (P_b, Q_b) is Bob's Public-Private Key pair. The key to be generated is called S.

- Alice Calculates $S_a = P_a \times Q_b \Rightarrow P_a \times P_b \times G$
- Bob Calculates $S_b = P_b \times Q_a \Rightarrow P_b \times P_a \times G$

Thus, it can be seen that $S_a = S_b$ and each side can produce the same secret key with this method is proved.

3.5.4 Public Key Recovery

The actual identity of the sender of the message is determined by using the Public Key obtained from the message and signature information. The transaction structure in the Ethereum network does not include the "from" field; the field is created virtually by the tools used to display transaction information, which is obtained using signature information. The general workflow of the ECRecover method is given in Figure 3.6.

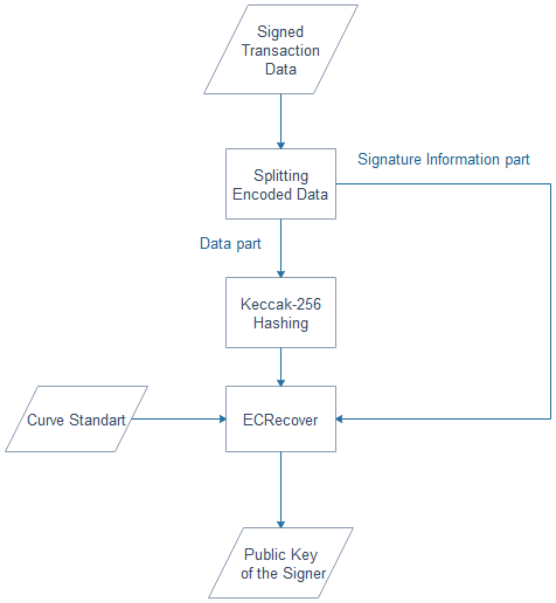

Figure 3.6 ECRecover UML diagram

In order to perform the public key recovery, the message hash, s and r values must be known. Since signature validation is used frequently in Ethereum blockchain, v value that is not added in classical signature processes is added to the transaction in order to accelerate the signature validation process. The general formula used for the public key recovery is given in Equation (3.2) (Antonopoulos & Wood, 2018).

$$Q = r^{-1} . (sR - zG) \tag{3.2}$$

Where;

- Q represents the public key of the sender of the message.
- r^{-1} is the mathematical inverse of the r-value of signature information.
- s is the signature value's "s" partition.
- The R-value is the ephemeral public key used in the ECDSA formula
- z value lowest n bits of hash of the transaction
- G value is generator point

There are two different values that provide the R-value according to the mathematics used in ECDSA. This also means there are two different possible public keys that can be generated. By checking whether the v value in the transaction is an odd or even number, it is understood which R-value should be used and thus the correct sender information is obtained. In the Ethereum network, this process is carried out by the ECRecover method, which is included in the libraries and smart contracts used.

3.6 Advanced Encryption Standart

Advanced Encryption Standard (AES) is a symmetric encryption algorithm used to ensure confidentiality during communication. The same key is used in the encryption and decryption stages in the symmetric encryption process, and this key must be known to each of the two communicating parties.

AES is referred to as a block cipher means that plain text with fixed lengths is converted to a ciphertext. In order to encrypt data of a size longer than this fixed length, the AES encryption operation is repeated. If the size of the text to be encrypted is smaller than 128 bits, the text size is expanded to 128 bits using padding.

The processes within this algorithm are separated into layers, and the data given as input is processed by all layers, respectively. Layers in AES are known as Key Addition Layer, Byte Substitution Layer, SiftRows Layer, and MixColumn Layer. The

process of processing the data covering all layers is called round. AES Encryption flow is illustrated in Figure 3.7.

AES algorithm can use the secret key of varying lengths such as 128/192/256/512 bits. The number of rounds varies on the secret key size and is calculated by the following Equation (3.3) (Dragomir & Vlad, 2017).

$$N_r = (K_s / 32) + 6 \quad (3.3)$$

Where;

- N_r: the number of rounds
- K_s: Key size

For example, when a secret key in 128-bit in size is used, AES performs encryption with ten rounds. Symmetric encryption algorithms such as AES generate an output ciphertext using an input plaintext and secret key information. No matter how many times the encryption process is repeated, the algorithm output does not change. This feature of symmetric algorithms is called the AES ECB mode for the AES algorithm and is considered unsafe because it is vulnerable to many attacks (Rogaway, 2011). This weakness of the ECB mode of the AES algorithm is exemplified by bitmap image encryption. Even if the bitmap is strongly encrypted with the 256-bit secret key using the AES algorithm, the image remains identifiable (Paar & Pelzl, 2010). Attacks in this mode are often exemplified using the Linux Tux image (Grace, 2015). In order to avoid this problem and potential attacks, an IV based encryption scheme is required to be used.

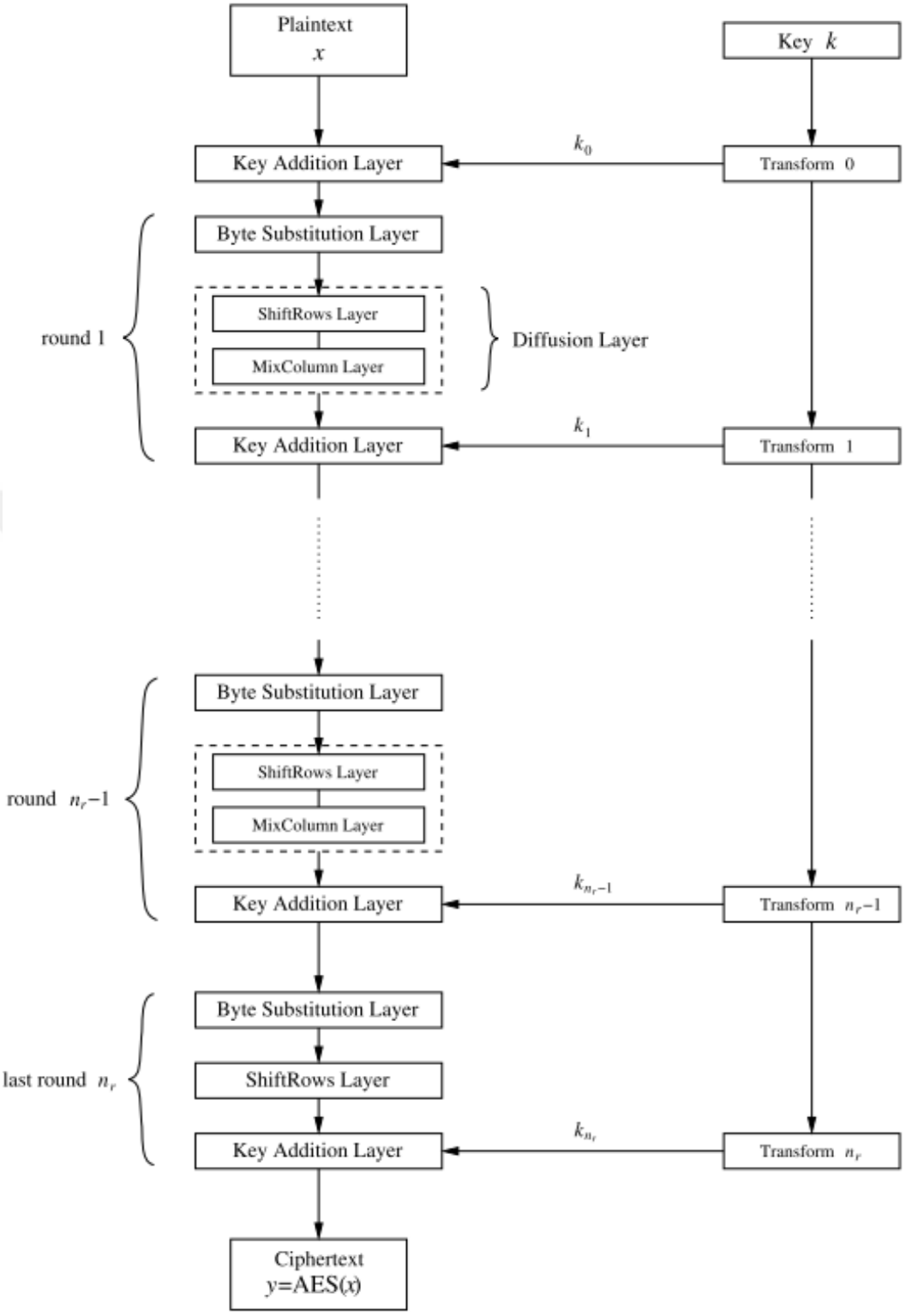

Figure 3.7 AES encryption flow diagram (Paar & Pelzl, 2010)

3.6.1 Cipher Block Changing (CBC) Mode

It is a probabilistic encryption scheme using an IV value. In this scheme, the random IV value must be selected before each encryption process. Also, in this mode, the encryption process of a block depends on all of the previous blocks (Rogaway, 2011). In this scheme, the result of a block and the plaintext input of the subsequent block are subjected to XOR operation. Since the first block does not have a block before it, XOR operation is performed by using the IV value given as the input parameter with the plaintext of the first block.

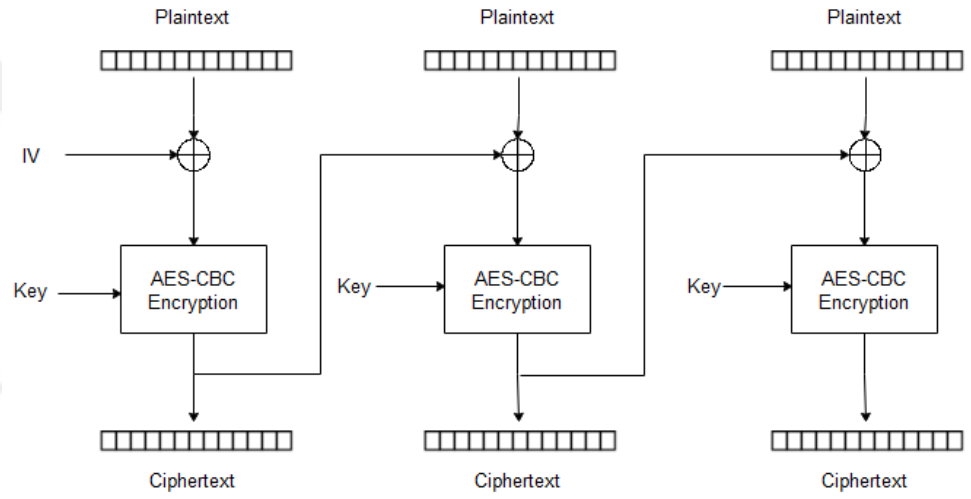

Figure 3.8 AES-CBC Encryption operation (Stallings, 2010)

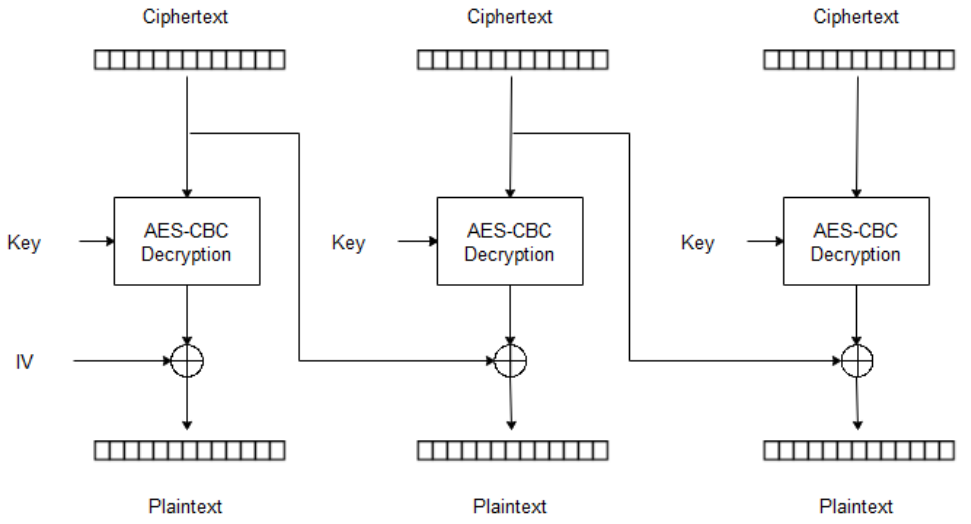

Figure 3.9 AES-CBC decryption operation (Stallings, 2010)

Encryption and decryption have different ways of working so that implementation errors can be avoided in design. The encryption scheme is illustrated in Figure 3.8, and the decryption scheme is illustrated in Figure 3.9. Furthermore, the CBC mode can provide message integrity. It is not necessary to keep the IV value secret in order to obtain an adequate level of security using CBC mode; the IV value can be sent over the network as plaintext. The IV value must be determined at unpredictable randomness for each plaintext to be encrypted (Dworkin, 2001). Otherwise, data encoded with CBC mode is the target of attacks such as chosen-plaintext attack as well as padding attack, which is based on exploiting the nature of the mode (Bard, 2006; Rogaway, 2011).

3.6.2 Counter (CTR) Mode

It is a nonce-based encryption scheme using IV. The IV value used in this scheme is referred to as a counter value because for each block converted to ciphertext, the numerical value in IV is incremented by one (Stallings, 2010). The IV value is 128 bits long. The first 96 bits are used as nonce values per stream, and the remaining 32 bits are used during the conversion of the current plaintext stream to ciphertext (Paar & Pelzl, 2010). The significant advantage of the CTR mode is that the encryption process can be paralleled, which means that in order to convert a block to ciphertext, it is not necessary to wait until the previous block has finished encoding. Moreover, since the IV value can be used as a nonce, the nonce value is sent only once at the beginning of the communication between the encrypted data transfer parties. Thus network bandwidth usage is reduced.

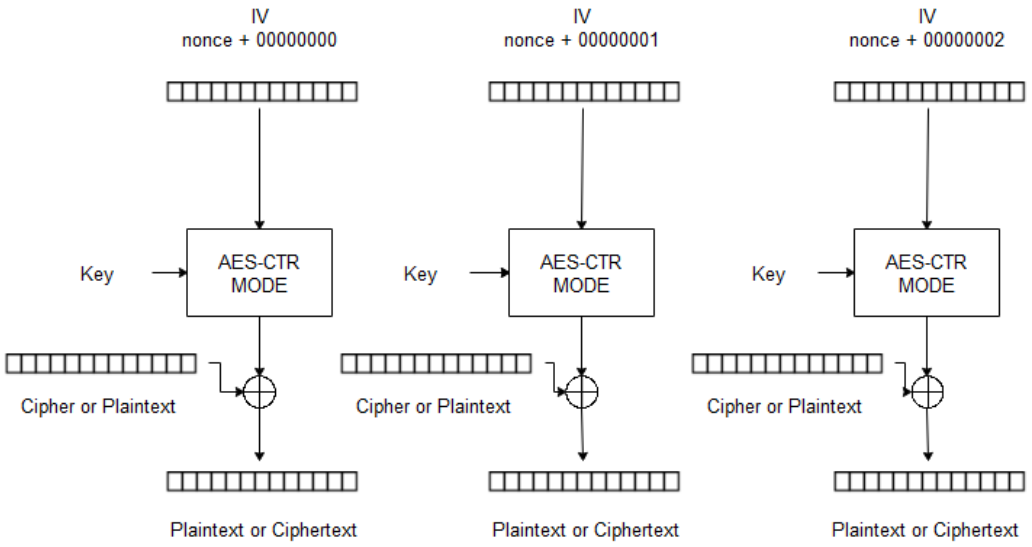

Figure 3.10 AES-CTR mode operation (Stallings, 2010)

In CTR mode, where encryption and decryption operations can be performed with a single method, the highest security breach comes from wrong implementation or wrong usage of the mode.CTR mode scheme is illustrated in Figure 3.10. In order to ensure that the data encrypted in CTR mode can be considered secure, the IV value used during the encryption operation must be unique for each block (Paar & Pelzl, 2010).

CHAPTER FOUR
IMPLEMENTATION

The main objective of this thesis is to be able to perform authentication and authorization with the blockchain network, which is believed to be a burdensome even not possible task by using cheaply available hardware and proving that high-security standards can be achieved for IoT network. The development environment consists of three main parts: thing, gateway, and blockchain authority nodes. Another significant point is to determine the product to be used in the application layer.

In this thesis, it is assumed that all devices included in this network must have a public & private key pair that is appropriate for the Ethereum Blockchain network. The pair can be generated offline within both the gateway and the thing devices, and since only the public key is needed to be shared by the devices, thus preventing the private key from being exposed is achieved. Moreover, the private keys of the devices are not known even by the system administrators in this way. Thus, the keys do not need to be stored, and a potential attack point is totally eliminated. When the method mentioned and this study are combined, a robust IoT infrastructure can be acquired.

Although there are developed light clients and modes for Ethereum Blockchain that are aimed to use on constrained environments, today it is still not possible to run these clients on constrained devices. Hence, the Ethereum Blockchain node cannot be hosted inside *things*. In this work, the thing communicates with the Ethereum network using the Ethereum JSON API placed in the Gateway. During this process, raw transaction signing and signature verification operations are performed in the thing to maintain the Trustless nature. In the application layer, MQTT is preferred since it is the most popular application layer protocol on the market. The preferred MQTT product uses the publish-subscribe communication pattern. In this thesis, the client-to-broker encryption scenario is used (HiveMQ, 2015).

After the authentication process, both the gateway and the *thing* derive the anonymous shared secret key with the ECDH algorithm, which they will use during

communication between each other. Anonymous shared key is derived from public&private key pair used to prove identity over the Etherem Blockchain network. Because of a different shared secret key is created for each communicating party, tampering a device within a topology that is based on our work cannot cause a security breach in the whole topology.

Pseudonymous communication is realized over the blockchain network, making it almost impossible for attackers to read the data until tampering any of the communicating parties. In this study, a high-availability authentication and authorization mechanism that resilient against failures and provides robust solutions to confidentiality and integrity problems, and also reduces the scalability problem is presented.

4.1 Experimental Setup

4.1.1 Thing Specification

ESP32 was chosen as the *thing* in this thesis. ESP32 is an easy-to-supply development kit and is available in a wide variety of versions. In this study, the ESP32-WROOM development kit with model code ESP32-D0WDQ6 was preferred. This development kit has a dual-core 240 Mhz processor, 512 kB Ram, and 4 MB external flash memory. This product has the features that the Internet of things devices should have, such as low power consumption. It also has hardware acceleration for RSA, ECC, AES and SHA methods. Thanks to all of these hardware features mentioned, ESP32 is able to perform transactions over the Ethereum Blockchain network.

During the software development phase for the ESP32 device, the PlatformIO IDE, which is aimed to use for IoT development and contains the all required components and features needed during development, is used (PlatformIO, 2014). C ++ is used as the programming language. The libraries used in the developed software are as follows.

- Firefly Wallet: The method developed in order to decode and convert raw transaction encoded with RLP received over the network is based on the code in Firefly Wallet (Moore, 2018).

- Web3E: The method of creating the object for the smart contract call or transaction sending, and encoding the generated object with RLP before being sent over the network is based on the code in the Web3E project (V. Zhang, 2018).

- Trezor Crypto: ECDSA, ECDH, ECRecover methods, secpk256k1 curve definitions, and Keccak-256 hashing method are taken from Trezor Crypto library (Rusnak, 2018).

- HWCrypto: With this library in the ESP32 Framework, it is possible to use AES-CBC and AES-CTR methods with hardware acceleration in ESP32 (Espressif, 2019).

- PubSubClient It is a frequently used and stable MQTT Client for ESP32. After the authentication process is done over the Blockchain Network, communication via the traditional network is done through this library (O'Leary, 2018).

4.1.2 Gateway Specification

The Raspberry Pi 3, which is the preferred gateway, has a four-core 1.2 GHz processor, 1 GB of RAM and built-in Wi-Fi and Bluetooth connectivity. The most significant advantage of this product is that it can run a full operating system. Thus, many software components used today can be adapted to work on the Raspberry Pi 3 product. The Raspbian, the official operating system for Raspberry Pi, is installed on the Gateway, and Javascript is used as the programming language during the program development process. Javascript has been preferred because it accelerates development processes. The products and libraries used in Gateway are as follows.

- NodeJS: This software is used as a Javascript Runtime Environment. This environment, which is preferred in many projects, has extensive library support (Heller, 2017).

- Mosca: It is an MQTT Server product that can be used as a standalone server or a module. Customization of Authentication and Authorization processes is supported by this product (Collina, 2017).

- Web3JS: Web3JS is a library with all the cryptographic and similar methods required to communicate on the Ethereum Network (Ethereum, 2016).

- MongoDB: It is a NoSQL database that serves as temporary object storage within the project. The records in MongoDB are in JavaScript object notation (JSON) format so that complex objects can be stored and read in MongoDB very easily (Chodorow, 2013). Although MongoDB is a persistent database, it can be used for temporary object storage thanks to its time-to-live(TTL) support (Baresse, 2018).

4.1.3 Blockchain Specification

During the development of the proposed authentication and authorization system, it was preferred to establish a Private Blockchain Network instead of using a blockchain development environment or using a blockchain test network in order to create a scenario that is as real-life as possible. The use of the Proof of Authority consensus algorithm is preferred, and the block creation time is set to 3 seconds on this network. The genesis configuration used in this study is given in Appendix 1. Go-Ethereum was used as the Ethereum client. There is no need to run a full node on the gateway, but the Go-Ethereum client is running in the light mode in the gateway to ensure that events sent over the smart contract are not counterfeit and prevent the creation of new attack points for malicious users. Three different authority nodes have

been added to this blockchain network. The technical specifications of these authority nodes are listed in Table 4.1.

Table 4.1 Blockchain authority nodes hardware specification

Location	Operating System	Client	CPU	RAM
USA	Ubuntu 16.04	go-ethereum	2.8 GHz(2 Core)	4 GB
Netherland	Ubuntu 16.04	go-ethereum	3.0 GHz(2 Core)	4 GB
Finland	CentOS 7.6	go-ethereum	3.4 GHz(4 Core)	4 GB

A smart contract is used to store the authorization list and perform authorization and access control of the device over the Blockchain Network.The source code of the smart contract that is created to use in this study is given in Appendix 2.

4.2 Proposed Blockchain-Based Authentication and Authorization System

This section describes the study step-by-step, which is authentication and authorization control over the Blockchain network between the *things* and the gateway. The UML diagram summarizing the overall study is shown in Figure 4.1

The authentication process starts with the creation of a smart contract for gateway by the system owner. This process is performed once when the *thing* is first turned on. After the created smart contract is defined on the gateway, the *things* on the system are added to the smart contract access control list by calling addOrModifyPermission function with the *thing's* address, resource identifier, and permission values. Permission defines the subject's authority to send or receive messages on the resource identifier. The sample transaction call is given in Appendix 4.

After defining devices with their permissions into the smart contract, all requirements for the authentication and confidential communication process between the gateway and the *thing* is met. All steps from the authentication process to the confidential communication process within the Blockchain and traditional networks are illustrated in Figure 4.1.

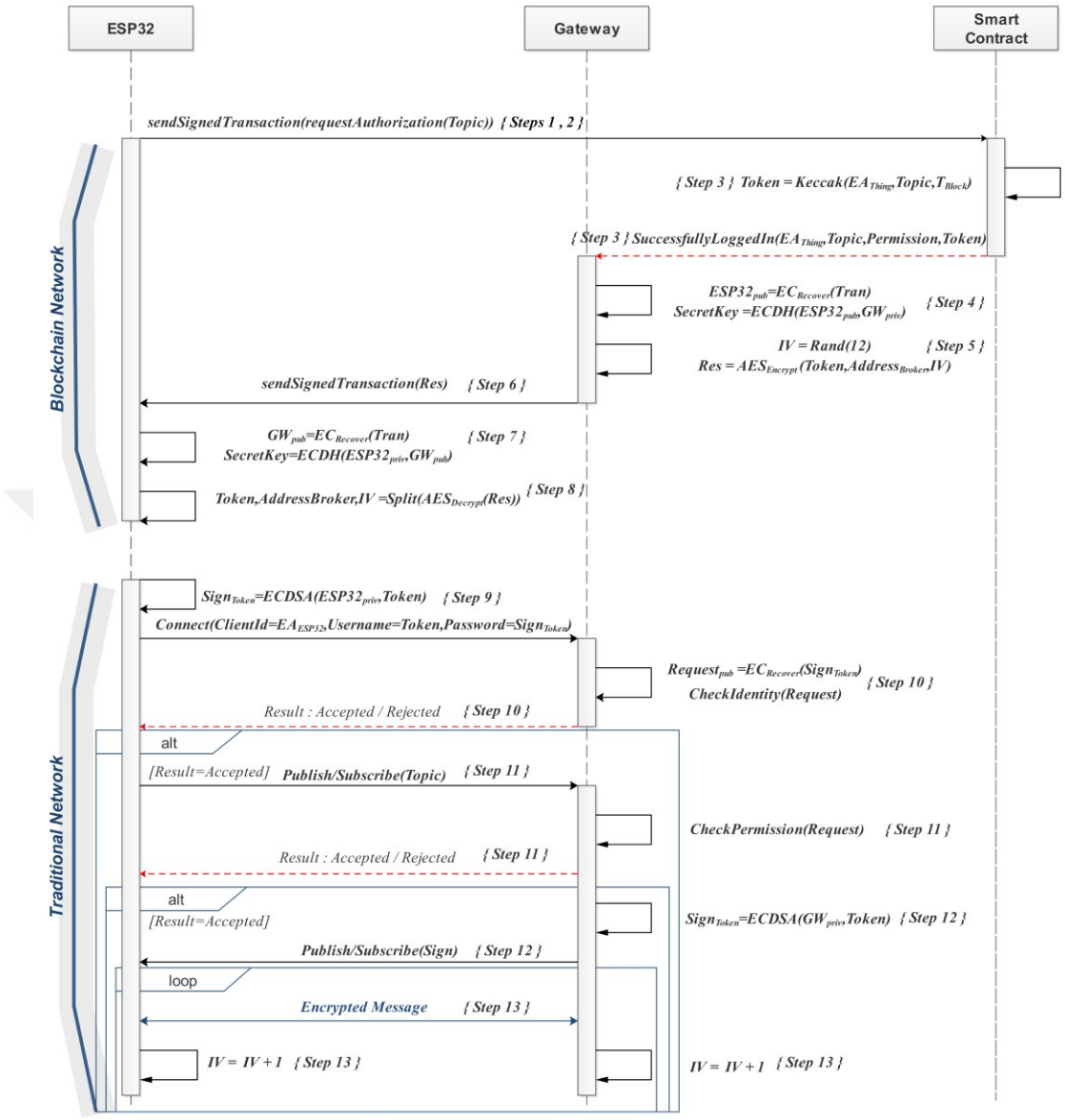

Figure 4.1 The UML diagram illustrates the study

Step 1. Before the *thing* starts the authentication process, it calls the AuthorizationRequest function by adding the resource identifier value. Since the data contained in the blockchain network is stored by all nodes within the network, this request is immediately answered by the local Ethereum node, which the request is sent. The call method is not sent to the blockchain network, and this method is only used to obtain read-only data for free. The permission over the resources information of the *thing* defined in the Smart Contract is collected in this way. This operation is repeated by the *thing* until an affirmative answer is received or predefined time is expired.

Step 2. After successfully detecting the existence of the authority of the *thing* on the resource identifier, it is ready to initiate the authentication and authorization process. The process proceeds as follows, started by the data prepared in the first step is written to the data property in the transaction structure, and after the remaining fields such as nonce, to are filled, the transaction is signed, and then the signed transaction is sent to the Smart Contract address. The sample transaction related to this step is given in Appendix 5.

Step 3. If the authorization is successful, the function on the smart contract creates a unique identifier by hashing the *thing's* Ethereum Address, resource identifier, and block timestamp with the Keccak-256 algorithm. Here the block timestamp value is used to obtain randomness. The function finally triggers the SuccessfullyLoggedIn event with the subject address, request identifier, permission, and unique identifier values. The function inputs and the function output is given in Table 4.2.

Table 4.2 Producing access token

Thing's Ethereum Address	0x540b21bE69A7AF3400209D61fc3E5D936C7e54D2
Resource Identifier	0x5465737453656e736f72 (TestSensor)
Block TimeStamp	1562016540
Keccak-256 Output (Token)	0xf2ba57ca165db275c8cd2966ef62b728a703f104346e45dec55c77fcef27bc1b

Step 4. The temporary storage preserves an access token that comes with the event triggered by the smart contract for a limited time. Ethereum addresses, as mentioned earlier, are not the public key. Gateway needs to find the corresponding transaction hash in the event to obtain the public key. It obtains *thing's* public key by using message hash and digital signature information in the raw transaction that triggers the event. Using the ECDH method with Gateway's private key and Thing's public key as parameters creates the secret key to be used in the symmetric encryption process anonymously. The resulting secret key is 32 bytes in size and is suitable for 256 bit ECC symmetric cryptographic algorithms. The sample inputs and outputs belong to this step are given in Table 4.3.

Table 4.3 Operations performed to obtain a secret key in the gateway

Used Method	Result	Output Data
getRawTransaction(Tx_{Hash})	Tx_{Raw}	0xf88a81e985051f4d5c0083419ce094fb595e72eed272b4aebb023c31ba0838cef0179780a4428b28a85465737453656e736f720000000000000000000000000000000000000000000001ca0f660cd561b7f93aaeeda2c5bfe62563206febd296954c0113d1726331e918433a0155b66870986bb67fc5f0f4f85311e3c0b1a4b4f823911bf14a88424075c2c5
getSenderPublicKey(Tx_{Raw})	$Thing_{Public}$	0x9d34f943435756dc6567d9e1d7526d2384798e1f9b469bf6d1a8fb7e14b008024c1d1aad03c9a78fdd1b972b0926bb1bae6d57939eb87575be932c6c5c56cca6
getSenderAddress(Tx_{Raw})	$Thing_{Ea}$	0x540b21be69a7af3400209d61fc3e5d936c7e54d2
ECDH ($GW_{Private}$,$Thing_{Public}$)	SecretKey	0x158e8440b886d7d6f2b270a1781ef68c792c7cf89719f8c797d0851bb91fd3db

Step 5. The gateway first generates the 12 bytes long random IV value to be used on the AES-CTR algorithm to obtain symmetric encryption on the conventional network. As stated in the previous section, the IV value is not a password and ensures that each data sent in encrypted form during communication on the conventional network is unique from each other, ensuring that communication tampering and replay attacks are prevented. Thereafter, the gateway concatenates the token, the MQTT broker address, and generated IV value then encrypts with the AES-CBC algorithm using the first 16 bytes of the transaction hash of the initialized request as the IV value. At the end of these steps, the gateway creates and sends a transaction to the *thing's* address with the generated information that is written the transaction's data property. The sample transaction the gateway created is given in Appendix 6.

Step 6. Transactions within a blockchain network are not indexed, so transactions to an address can only be obtained by scanning the inside of all blockchain blocks. This process is costly in terms of time and energy and is unnecessary to be performed by *things*. On behalf of the *thing*, this work is done by the developed proxy node. Finding a received transaction is done as follows: The *thing* calls the

ext_SearchTransaction method inside the proxy node via JSON-RPC, which is located in the Gateway device, with the transaction hash information it sends it is own Ethereum address and address of the Smart Contract,By doing this, the *thing* can check the presence of a transaction that is sent on it after certain transactions. The source code of the developed JSON-RPC proxy is given in Appendix 3. This search process is repeated until either the transaction hash information is found by the method or predefined maximum allowed time interval is exceeded. A sample call is given in Figure 4.2.

```
{ jsonrpc: '2.0',
  method: 'ext_searchTransaction',
  params: { from: '0xF18388Bc327F2aeE6712551F52E95BE8a8ac2072',
            to: '0x540b21bE69A7AF3400209D61fc3E5D936C7e54D2',
            startBlockNumber: 1259845
            count: '1' }
}
{ jsonrpc: '2.0',
   result: '0x52f00ea569d9190c3adb579341f68a2fb13f1cca9de5ad3ebaa3
58e2a3e6b857'
}
```

Figure 4.2 The RPC call to find the received transaction

Step 7. Once the *thing* discovers the transaction hash sent by the gateway, the *thing* gets the raw transaction content using the getRawTransaction method over the blockchain network via the Ethereum light client node hosted inside the Gateway to retrieve transaction content and verify the signature. In the next step, the hash information is generated to verify the signature information placed in the transaction. In order to perform a verifying signature task, it is necessary to divide the received raw transaction into length, content, and signature parts. The content part is hashed by the Keccak-256 algorithm. Then this hash and the signature part are used on the ECRecover algorithm as input parameters to obtain the public key. The resulting public key is not an Ethereum address, and since the *thing* only knows the Ethereum address, the Ethereum address needs to be derived from the public key. In order to get the Ethereum address, the resulting public key is hashed by applying the Keccak256

algorithm, and the last 20 bytes are taken. With this process, the Ethereum address that sends the message is determined. After this, it generates a symmetric key using the ECDH algorithm with its own private key and gateway's public key, which is derived from the transaction. As a result, the *thing* acquired the same key as the gateway anonymously. The methods and the outputs used in this step are given in Table 4.4.

Table 4.4 Operations performed to obtain a secret key in the *thing*

Used Method	Result	Output Data
getRawTransaction(Tx_{Hash})	Tx_{Raw}	0xf8eb8201308502540be40082c35094540b21be69a7af3400209d61fc3e5d936c7e54d2843b9aca00b8803438313931646132656561326364333565343732656135623961333862383138323936333366633932313631313364626331373930326134316531613235343663353662643263353665356133313464663538373230363132396363633436316631306463633434303333386333643236643665336234356338396237323439 1ba079b02a3d44738f1c6855e993064c9006c3667ff14753983a95ac7053432edf79a028c2bfd40f0401b63d7acc8d19f62d636ba4b04a04f9fcbb633f2d267ff35149
getSenderPublicKey(Tx_{Raw})	GW_{Public}	0x50cc4e4ada2503527d7632119b3d06d8a23d5a7fc452e8873d2bf94ab5f3dc90a0a41367fcd1a054c2a75450d48b1c45f54f44cebf29af872f52f8673e5826e6
getSenderAddress(Tx_{Raw})	GW_{Ea}	0xfb595e72eed272b4aebb023c31ba0838cef01797
ECDH ($Thing_{Private}$,GW_{Public})	SecretKey	0x158e8440b886d7d6f2b270a1781ef68c792c7cf89719f8c797d0851bb91fd3db

Step 8. The imported Raw Transaction is in the RLP Encoded form. For the data sent by the gateway to be read, the transaction is needed to be decoded and parsed into a structure by the *thing*. In this transaction, the data property is encrypted on the gateway as a necessity of pseudonymous communication. This data property is decrypted using the shared secret key generated in the previous step and IV value. The IV value is the first 16 bytes of the transaction hash of the initialized request. With the decoded message, the *thing* obtains required values that are the token to be used for the connection, the MQTT Broker address, and the nonce value to be used as IV during

point-to-point communication on the traditional network. With this step, the steps need to be taken over the Blockchain network is completed, and the *thing* is ready to log on with the required information it gathered to the MQTT broker.

Step 9. The *thing* uses the Ethereum Address as the Client ID and the token from the transaction to connect to the MQTT Broker as the username. The *thing* signs the token which receives from the gateway with ECDSA Algorithm using its own Ethereum private key. It puts the signature information to the password field. A sample data prepared by the *thing* is given in Table 4.5.After these steps, the *thing* sends the connection request onto the MQTT Broker.

Table 4.5 Example MQTT connection parameters

Method	Input Parameter	Parameter Value
Connect	Client Id	0x540b21bE69A7AF3400209D61fc3E5D936C7e54D2
	Username	0x3fefe513d309972ab45c663915a476f57adc23f0427c686da5ba22faf86adb5
	Password	0xf7d57b557931524730a5f78e13b3a1d9a44f192e1e65b906361a05988452cf07710e76a5a2dbee2147d934637232669dc9fde581c1a8c85eae841620faee07c21b

Step 10. The MQTT broker searches the client ID and token information in the temporary storage. If there is no such information in the temporary storage, it rejects the connection request. If the data exists, the public key is obtained by using the token sent in the username field and the signature information sent in the password field as input for the ECRecover method, after this key is converted to the Ethereum address, it performs verification by comparing it with the information in the database. If verification is successful, the broker stores the resource identifier and permission information in its in-memory session storage and the authentication process will be completed.

Step 11. In MQTT Protocol, resource identifiers are called topics. The permission variable is used to check whether the *thing* can be a subscriber or publisher. The *thing* sends a publish or subscribe request to the broker for a topic. MQTT Broker checks the authority of the *thing* by looking at resource identifiers and permission records associated with the *thing* that stored on MongoDB, which is used as temporary session

storage. If authorization is a success, then the access control phase is completed. Otherwise, the connection is dropped.

Step 12. In order to provide mutual authentication, an identity of the broker must be verified by the *thing*. The MQTT protocol does not provide a mechanism to serve that purpose except the TLS protocol that works on the network layer. Therefore, this process is needed to be done on the application layer side. Since MQTT is a protocol with a publish-subscribe communication pattern, no response can be sent over the application layer after the publish request. To achieve this purpose, the *thing* sends the subscription request to the MQTT broker. The MQTT broker understands that the request is linked to the main topic of the Mutual Authentication phase and accepts the request. Signs the token information via the private key it owns and publishes it through the topic to which the *thing* subscribed to.

Step 13. After verification of the signature on the subscribed topic is done by the *thing*, the mutual authentication process is completed, and communication is started. The *thing* encrypts each packet it willing to publish using the AES-256-CTR algorithm with the secret key generated and the IV value is sent by the gateway. After each published message, the value of IV is increased by one to ensure that each encrypted message is almost unique, and replay attacks are prevented.

CHAPTER FIVE
ANALYSIS OF THE PROPOSED MECHANISM

In this chapter, the security analysis of the proposed method and the performance analysis of the devices used are given in different sections.

5.1 Security Analysis

5.1.1 Confidentiality

The entire access control list is stored in the blockchain network. Even if the variables containing the access control list in the smart contract are defined as private, this list can be obtained by scanning the transaction log or reverse-engineering from the database. However, 256-bit end-to-end encryption using the key, which is created per communication parties, anonymously generated during the authentication process via the MQTT protocol on the traditional network is achieved with this proposed method. Some types of attacks that threaten confidentiality are discussed below.

5.1.1.1 Man-in-the-Middle Attack

In this work, confidentially is achieved by providing encrypted, pseudonymous communication between two endpoints with unique Ethereum addresses on the network that can only be solved by these endpoints over both the blockchain and traditional network. The secret key used during encryption is 256 bits in size. This encryption is so secure that it is used by the NSA to transmit confidential information. During the authentication process, the attacker can intercept the connection between the *thing* and the gateway and sends its own public key. In this case, the shared key will be generated by both the *thing* and the gateway using the public key of the attacker; thus, all communication can be resolved by the attacker. As a first precaution against this situation, each IoT device compares the public key to the Ethereum key during the authentication process. In addition, the address of the gateway Ethereum that the *thing* willing to connect is defined inside the *thing* statically.In addition, the

gateway hosts the local Ethereum light node, and the accuracy of the data is provided by this node. Thus, blockchain communication during the access control process cannot be tampered with.

5.1.1.2 Node Capturing Attack

In this study, one unique shared secret is generated for each gateway and thing pair exchanging data, using the Elliptic Curve Diffie-Helman (ECDH) algorithm. Even if the connected *things* is captured, it is not possible for attackers to decode communication within overall the IoT network.

If only the Gateway is captured, communication between the gateway and all *things* connected to the gateway can be resolved, but it is still not possible to seize sensitive information such as the private key of the *things*. At the same time, it is not possible to damage the blockchain network used for authentication and Access control by capturing the gateway.

The preferred Proof of Authority consensus algorithm in the Blockchain network works with the logic of creating the blocks with fixed times and validating transactions by signing instead of finding solutions to complex mathematical problems. When the node-capturing attack on the authority node is the case,since all processes in the protocol require the signature of 51% of the authority nodes, the attacked node can be removed from the Blockchain network with the majority of authority nodes that remains. Therefore, the security of the blockchain network is directly proportional to the number of authority nodes and the geographically distributed distribution of these nodes.

5.1.1.3 Eavesdropping Attacks

In the computer networks, eavesdropping attacks are carried out by the attacker sniffing and recording the communication between two nodes via specialized software and hardware. In this study, data is not sent in plain text during communication

between two nodes. AES-CBC algorithms are preferred for data transmission over the blockchain network and the AES-CTR algorithm for data transmission over the traditional network. The shared secret key is created explicitly by each of the two nodes to be used only in two nodes of communication without network communication with the ECDH algorithm. This switch can only be obtained by physically capturing one of the nodes. In the encrypted text sent with the IV (nonce) values used during the encryption operation, randomness was added, and the data pattern was wholly prevented from being found by the attackers.

5.1.2 Authentication and Integrity

The tamper-proof Smart Contract generates all event messages. To prevent replay attacks, token inside an event created with block timestamp value to create randomness. Raspberry Pi receives event messages via the Ethereum light client that is contained in it, so that fake event messages cannot be sent over the network. ESP32 and Pi decode and decrypt the full raw transaction message in order to verify the identity of the sender and reading the data inside the transaction. Even if attackers change the content of the data, it is not possible to create a proper signature of the changed data. In this way, no predefined trusted sources are needed for the system to operate, and the system is secured against replay attacks.

All data on the blockchain network, including access control lists, are signed and validated by authoritative nodes and stored in the form of the Merkle tree data structure. Therefore, it is almost impossible to modify the access control list by attackers.

5.1.2.1 Modification Attack

The validity and integrity of the access control list are guaranteed by the blockchain structure. In addition, the whole process of authorization control is managed by smart contracts, and an event is created as a result of a successful authentication process. In this study, there is an Ethereum light node in the Gateway, and the gateway receives

event messages from this node. Since the gateway does not communicate over the network to read event messages, it is not possible to change event messages by opponents to allow unauthorized *things* to log on to the gateway.

5.1.2.2 Fabrication Attack

On the access control smart contract within the blockchain network, only the Ethereum address defined in the admin role on this contract can add and remove authorization for *things*. In addition, the proposed robust authentication method for the MQTT broker is to make it impossible to log in with a credential that is not on the smart contract inside the blockchain network.

5.1.2.3 Replay Attack

In this study, authentication replay attacks on the gateway have been wholly prevented. In the blockchain network, a variable is stored for each address, called nonce, which represents the number of messages sent by the address. Transactions with a previously used nonce value are rejected in the blockchain network. Therefore, for each transaction to be sent, this value is incremented, added to the transaction packet and uniqueness per message is created. The AES-CTR algorithm used for encryption on the traditional network has a counter value. Both nodes in communication increase their counter value by one after successful encryption & decryption. Even if the repeated message is sent on the gateway, the encrypted message cannot be resolved by the gateway, and become invalid because the counter value used to encrypt this message is no longer used by the gateway. With this, replay attacks are prevented.

5.1.2.4 Unauthorized Access

Thanks to the authorization module developed for MQTT Broker, the authenticated *thing* on the gateway can only subscribe or publish to the resource identifiers defined in the smart contract. The gateway accepts or rejects the connection request by

checking the resource identifier and permission values received from the event that it stores in MongoDB, and thus preventing the connection to the identifiers where the *things* are not authorized is achieved.

5.1.2.5 Sybil Attack

In this study, the Ethereum private key is used as an identity certificate. Messages sent to the Blockchain network and logon requests forwarded to the broker are signed with this key. It is not possible to obtain any credentials by listening to the network by malicious nodes within this network.

5.1.3 Availability

In this blockchain-based study, there is no single-point-of-failure point for the authentication process. All nodes within the created the Blockchain network store smart Contract data, which contains the access control list. The overall authentication and access control system is therefore robust and resilient to hardware failures, data loss, and distributed denial of service (DDoS) attacks.

5.1.3.1 Gateway Malfunction

In this study, to eliminate the need for the trusted third party, the gateway address is permanently defined inside the *thing*. In the case of a gateway fault, the address of the new gateway must be set inside the *thing*. The NDEF protocol can be used in the definition process. A fault rescue mechanism for defining the new Ethereum address of the Gateway is out of the scope of this study.

5.1.3.2 Communication Desynchronition

AES-CTR algorithm is used for secure data transfer between the *thing*, and the gateway on the traditional network requires a nonce-based IV value that sequentially increased on each operation. That is, the IV value, which is determined by the gateway

in the authentication process, is increased by one for each message sent and transmitted from the *thing* over the MQTT protocol. If the message sent by the *thing* is lost, only the counter value held by the *thing* will increase, and the synchronization will be lost. This situation can be prevented by using QoS 1 or QoS 2 mode while sending messages over the MQTT protocol.

5.2 Performance Analysis

The primary purpose of this study is to offer an authentication mechanism over the Blockchain Network that is aimed to use on constrained-resource devices. Hence, performance tests are concentrated on the ESP32 device, which is preferred as a constrained-resource device.

In order to obtain the most accurate quantitative performance results during the measurements, each algorithm performance used in the process is measured by running 1000 repetitions, and overall performance measurements are gathered with 100 repetitions. CPU Time is taken by using the getCycleCount method in the ESP32 Framework.

All ECC algorithms run smoothly and flawlessly by ESP32. A hash of a raw transaction data can be encrypted with the ECDSA algorithm within 23 milliseconds. It takes 82 milliseconds for creating the secret key to be used during data communication between two endpoints with the ECDH algorithm. The ECRecover algorithm that is used to validate the signature and create a secret key is the most CPU-consuming process and takes 197 milliseconds to complete. These results are illustrated in Figure 5.1

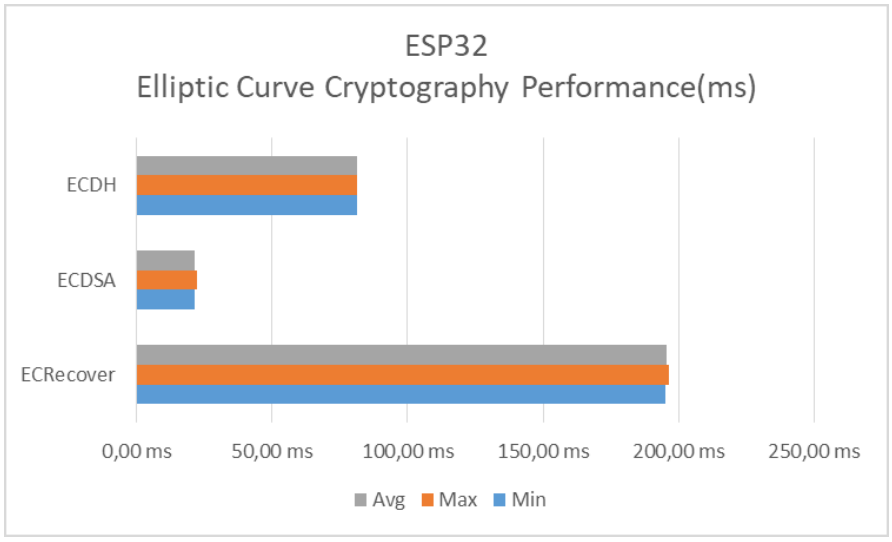

Figure 5.1 Elliptic curve cryptography performance of ESP32

Encoding, decoding, and hashing methods do not take more than 15 milliseconds in this work. Algorithms and its completed times are shown in Figure 5.2

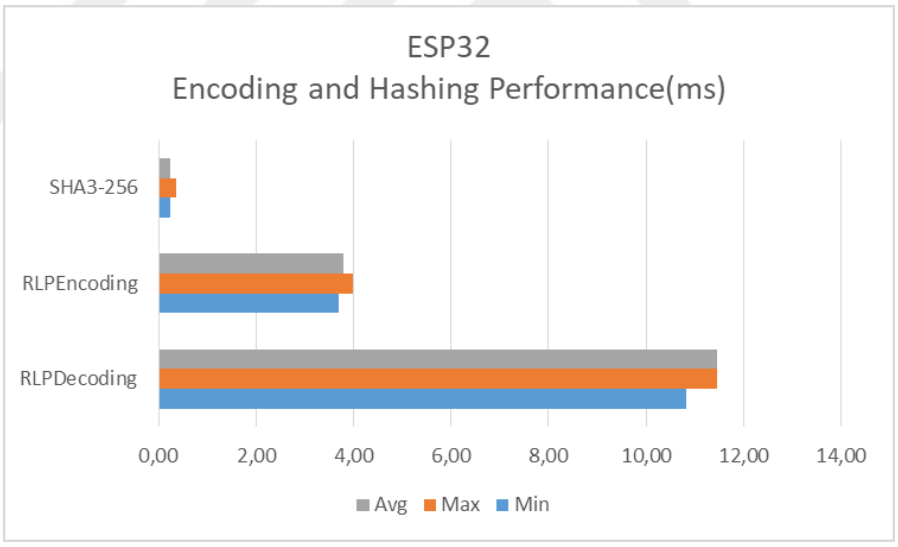

Figure 5.2 Encoding and hashing performance of ESP32

All transaction-related steps are directly proportional to blockchain creation time. Proof of Authority consensus algorithm uses predefined block-creation time. In this work, the time is set to 3 seconds. Three different nodes are set up on geographically distributed located To create a more realistic blockchain environment. The authentication process performed by ESP32 was examined in three phases as sending login requests, the response received, response parsing, respectively. The login request

phase is completed in an average of 2 seconds, the response received phase, which represents the Blockchain network performance is completed in an average of 9 seconds, response parsing phase is completed in an average of 2 seconds. The complete access control and authorization processes on the Blockchain network are completed on an average of 13. These results are illustrated in Figure 5.3

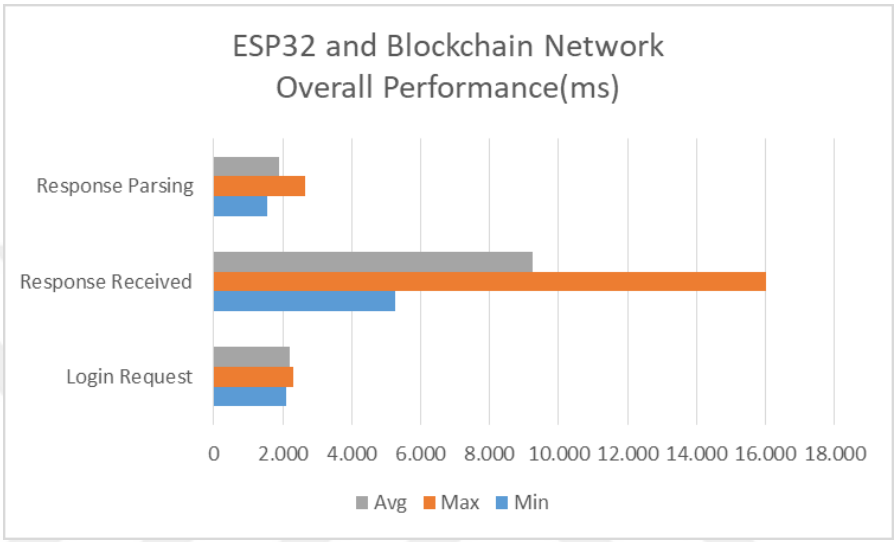

Figure 5.3 Authentication performance of ESP32

CHAPTER SIX
CONCLUSION AND FUTURE WORK

To summarize, the current usage areas of the Internet of things and the ongoing security issues are explained. The importance of authentication and authorization is emphasized with examples. How blockchain technology can be used to solve problems that are still up to date for the Internet of things has been explained by highlighting the unique features of the blockchain.

In many studies, either the *things* do not use the blockchain network, or the device is chosen as a *thing* is not suitable for use in real-life scenarios, such as the Raspberry Pi, which has high energy consumption and high processor speed. Opposite of these studies, ESP32 is selected as a *thing* to show the usability of our study on a real-life scenario. It has been shown that all cryptographic algorithms used in this study can run flawlessly on ESP32. By using the hardware acceleration support of ESP 32, a Keccak-256 hash can be signed with the secp256k1 curve in an average of 23 milliseconds. Elliptic curve Diffie-Helman algorithm with the secp25k1 curve takes an average of 82 milliseconds to complete.

In addition to providing authentication and authorization of *things* via the blockchain, pseudonymous communication between the *thing* and the gateway is presented in a transaction initiated on the blockchain, which is a public distributed ledger. Thus, it is shown that confidentiality can be provided over the blockchain network.

In this study, confidentiality is provided in both the Blockchain and the traditional network using AES algorithm in different modes. To solve the secret key sharing problem and to ensure the key is not seized, the secret key is created anonymously for each communication party by using their public and private key pair that they used on the Blockchain network. Integrity is provided by creating a robust authentication and authorization mechanism by using the smart contract that runs over the Blockchain network. The entire access control list is conserved inside the Blockchain network, and

thanks to the tamper-proof structure of the Blockchain, the access control list can not modified by malicious users. The proposed mechanism can work on unlimited devices, and these devices can be placed geographically distributed. Thus, the single-point-of-failure problem does not occur with the proposed mechanism, and high availability for authentication and authorization process is achieved.

Furthermore, since MQTT does not provide any security mechanism aside TLS to secure communication between client and broker, the client-to-broker payload encryption model which uses anonymously created secret key is applied to communication to strengthen the security of MQTT broker is provided.

In future work, a complete authentication and authorization architecture that eliminates assumptions made in this study will be developed.

REFERENCES

Aazam, M., Hung, P. P., & Huh, E. N. (2014). Smart gateway based communication for cloud of things. *IEEE ISSNIP 2014 - 2014 IEEE 9th International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Conference Proceedings*, (April), 1–6.

Abomhara, M., & Koien, G. M. (2014). Security and privacy in the internet of things: current status and open issues. *2014 International Conference on Privacy and Security in Mobile Systems, PRISMS 2014 - Co-Located with Global Wireless Summit*, 1–8.

Al-Sarawi, S., Anbar, M., Alieyan, K., & Alzubaidi, M. (2017). Internet of things (IoT) communication protocols: review. *ICIT 2017 - 8th International Conference on Information Technology, Proceedings*, (October), 685–690.

Almadhoun, R., Kadadha, M., Alhemeiri, M., Alshehhi, M., & Salah, K. (2018). A user authentication scheme of IoT devices using blockchain-enabled fog nodes. *Proceedings of IEEE/ACS International Conference on Computer Systems and Applications, AICCSA*, 1–8.

Aloul, F., Zahidi, S., & El-Hajj, W. (2009). Multi factor authentication using mobile phones. *International Journal of Mathematics and Computer Science*, *4*(2), 65–80.

Amara, M., & Siad, A. (2011). Elliptic Curve Cryptography and its applications. *7th International Workshop on Systems, Signal Processing and Their Applications, WoSSPA 2011*, 247–250.

Amrutiya, V., Jhamb, S., Priyadarshi, P., & Bhatia, A. (2019). Trustless two-factor authentication using smart contracts in blockchains. *2019 International Conference on Information Networking (ICOIN)*, 66–71.

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

Arvanaghi, B. (2018). *Explaining the genesis block in Ethereum –*. Retrieved July 27, 2019, from https://arvanaghi.com/blog/explaining-the-genesis-block-in-ethereum/

Asim, M. (2017). A survey on application layer protocols for Internet of things (IoT). *2017 Seventh International Symposium on Embedded Computing and System Design*, *8*(3), 996–1000.

Auth0. (2018). *Authenticating & authorizing devices using MQTT with Auth0*. Retrieved July 26, 2019,from https://auth0.com/docs/integrations/authenticating-devices-using-mqtt

Babaeizadeh, M., Bakhtiari, M., & Mohammed, A. M. (2015). Authentication methods in cloud computing: a survey. *Research Journal of Applied Sciences, Engineering and Technology*, *9*(8), 655–664.

Badretdinov, T. (2018). *How to create an Ethereum wallet address from a private key*. Retrieved July 14, 2019, from https://www.freecodecamp.org/news/how-to-create-an-ethereum-wallet-address-from-a-private-key-ae72b0eee27b/

Bard, G. V. (2006). A challenging but feasible blockwise-adaptive chosen-plaintext attack on SSL. *SECRYPT 2006 - International Conference on Security and Cryptography, Proceedings*, 99–109.

Baresse, L. (2018). *Auto expire documents with MongoDB*. Retrieved August 13, 2019, from https://medium.com/@Baresse/auto-expire-documents-with-mongodb-772fee84cc23

Bayer, D., Haber, S., & Stornetta, W. S. (1993). Improving the efficiency and reliability of digital time-stamping. *Sequences II*, 329–334.

Bella, G., Massacci, F., & Paulson, L. C. (2003). Verifying the set registration protocols. *IEEE Journal on Selected Areas in Communications*, *21*(1), 77–87.

Bitrates. (2018). *What is EVM (The Ethereum Virtual Machine)?* Retrieved August 23, 2019, from https://www.bitrates.com/guides/ethereum/what-is-the-

unstoppable-world-computer

Bonneau, J., Herley, C., van Oorschot, P. C., & Stajano, F. (2015). Passwords and the evolution of imperfect authentication. *Communications of the ACM*, *58*(7), 78–87.

Braeken, A. (2018). PUF based authentication protocol for IoT. *Symmetry*, *10*(8), 352.

Bressler, M. S., & Bergen, C. W. Von. (2014). Never Underestimate the Power of a Backhoe: Integrating Single Points of Failure into Strategic Planning. *American Journal of Management Studies*, 1(1), 1-22.

Bryk, A. (2018). *Entropy as a Service – increasing the security of your encryption*. Retrieved July 14, 2019, from https://www.apriorit.com/dev-blog/535-entropy-as-a-service

Buterin, V. (2013). Ethereum white paper. *GitHub Repository*, 22–23.

Buterin, V. (2015). *Merkling in Ethereum*. Retrieved July 28, 2019, from https://blog.ethereum.org/2015/11/15/merkling-in-ethereum/

Buterin, V. (2016). *Simple replay attack protection · Issue #155*. Retrieved August 4, 2019, from https://github.com/ethereum/EIPs/issues/155

Chodorow, K. (2013). *MongoDB: the definitive guide: powerful and scalable data storage*. Sebastopol: O'Reilly Media.

CIA. (2016). *Country comparison :: electricity - consumption — the world factbook - central intelligence agency*. Retrieved July 28, 2019, from https://www.cia.gov/library/publications/resources/the-world-factbook/fields/253rank.html

Collina, M. (2017). *Mcollina/mosca: MQTT broker as a module - GitHub*. Retrieved June 4, 2019, from https://github.com/mcollina/mosca

Consensys. (2016). *Ethereum, gas, fuel*. Retrieved July 22, 2019, from https://media.consensys.net/ethereum-gas-fuel-and-fees-3333e17fe1dc

Dias. (2017). *The 5 factors of authentication*. Retrieved July 14, 2019, from https://medium.com/@renansdias/the-5-factors-of-authentication-bcb79d354c13

Digicert. (2017). *Symantec backs its CA*. Retrieved July 20, 2019, from https://community.digicert.com/en/blogs.entry.html/2017/03/24/symantec-backs-its-ca.html

Dinur, I. (2015). Security evaluation of SHA-3. Retrieved July 10, 2019, from https://www.cryptrec.go.jp/estimation/techrep_id2402.pdf.

Dragomir, I. R., & Vlad, A. (2017). Statistical testing of the initializing stage of a block cipher, as part of the security assessment. *UPB Scientific Bulletin, Series A: Applied Mathematics and Physics*, *79*(2), 207–222.

Dworkin, M. J. (2001). Recommendation for block cipher modes of operation. Retrieved July 28, 2019, from https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38c.pdf

Espressif. (2019). *Hwcrypto · espressif/arduino-esp32*. Retrieved August 14, 2019, from https://github.com/espressif/arduino-esp32/tree/master/tools/sdk/include/esp32/hwcrypto

Ethereum. (2016). *Web3.js - Ethereum javascript API*. Retrieved August 1, 2019, from https://web3js.readthedocs.io/en/v1.2.1/

Farooq, M. U., Waseem, M., Khairi, A., & Sadia, M. (2015). A critical analysis on the security concerns of Internet of things (IoT). *International Journal Of Computer Applications*, *111*(7), 1–6.

Fenton, J. L., Newton, E. M., Perlner, R. A., Regenscheid, A. R., Burr, W. E., Richer, J. P., … Burr, W. E. (2017). Digital identity guidelines. *NIST Special Publication 800-63B*, 2–79.

Gilsenan, C. (2018). *SMS: The most popular and least secure 2FA method*. Retrieved July 20, 2019, from https://www.allthingsauth.com/2018/02/27/sms-the-most-popular-and-least-secure-2fa-method/

Gour, R. (2018). *Top 10 applications of IoT*. Retrieved July 20, 2019, from https://dzone.com/articles/top-10-uses-of-the-internet-of-things

Grace, Z. (2015). *Attacking ECB*. Retrieved August 10, 2019, from https://zachgrace.com/posts/attacking-ecb/

Gupta, A., Christie, R., & Manjula, R. (2017). Scalability in Internet of things: features, techniques and research challenges. *International Journal of Computational Intelligence Research*, *13*(7), 1617–1627.

Harsha, M. S., Bhavani, B. M., & Kundhavai, K. R. (2018). Analysis of vulnerabilities in MQTT security using Shodan API and implementation of its countermeasures via authentication and ACLs. *2018 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2018*, 2244–2250.

Haughn, M.; Giblisco, S. (2014). *What is confidentiality, integrity, and availability (CIA triad)?* Retrieved July 15, 2019, from https://whatis.techtarget.com/definition/Confidentiality-integrity-and-availability-CIA

Heller, M. (2017). *What is Node.js? The javascript runtime explained.* Retrieved August 14, 2019, from https://www.infoworld.com/article/3210589/what-is-nodejs-javascript-runtime-explained.html

Hertig, A. (2017). *Ethereum spam attacks are back – this time on the test network*. Retrieved July 13, 2019, from https://www.coindesk.com/ethereum-spam-attacks-back-time-test-network

Hitchens, R. (2018). *Calls vs. transactions in Ethereum smart contracts*. Retrieved August 14, 2019, from https://blog.b9lab.com/calls-vs-transactions-in-ethereum-smart-contracts-62d6b17d0bc2

HiveMQ. (2015). *MQTT security fundamentals – securing mqtt systems*. Retrieved June 28, 2019, from https://www.hivemq.com/blog/mqtt-security-fundamentals-payload-encryption/

Hollander, L. (2019). *The Ethereum Virtual Machine — how does it work?* Retrieved July 20, 2019, from https://medium.com/mycrypto/the-ethereum-virtual-machine-how-does-it-work-9abac2b7c9e

Hoyos, A. (2017). *May 31, 2017 Security Incident (UPDATED June 8, 2017) - OneLogin*. Retrieved July 20, 2019, from https://www.onelogin.com/blog/may-31-2017-security-incident

Hughes, C. (2015). *Access management: centralized vs. distributed - advanced software products group*. Retrieved July 20, 2019, from http://aspg.com/access-management-centralized-vs-distributed/#.XTKrGegzaUk

Karagiannis, V., Chatzimisios, P., Vazquez-gallego, F., & Alonso-zarate, J. (2015). A survey on application layer protocols for the internet of things. *Transaction on IoT and Cloud Computing*, *3*, 11–17.

Kasireddy, P. (2017). *How does Ethereum work, anyway?* Retrieved July 28, 2019, from https://medium.com/@preethikasireddy/how-does-ethereum-work-anyway-22d1df506369

Kenneth, H. (2018). *Ethereum account*. Retrieved July 26, 2019, from https://medium.com/coinmonks/ethereum-account-212feb9c4154

Khan, M. K., & Zhang, J. (2007). Improving the security of "a flexible biometrics remote user authentication scheme." *Computer Standards and Interfaces*, *29*(1), 82–85.

Kim, K. (2018). *Modified Merkle Patricia trie — how Ethereum saves a state*. Retrieved July 19, 2019, from https://medium.com/codechain/modified-merkle-patricia-trie-how-ethereum-saves-a-state-e6d7555078dd

Kolias, C., Kambourakis, G., Stavrou, A., & Voas, J. (2017). DDoS in the IoT: Mirai and other botnets. *Computer*, *50*(7), 80–84.

Lamport, L., Shostak, R., & Pease, M. (1982). The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3),

382-401.

Latif, L. (2011). *Comodo admits hackers issued fraudulent SSL certificates*. Retrieved July 20, 2019, from https://www.theinquirer.net/inquirer/news /2037113/comodo-admits-hackers-issued-fraudulent-ssl-certificates

Levi, A., & Caglayan, M. U. (1997). The problem of trusted third party in authentication and digital signature protocols. In *the Proceedings of the Twelfth International Symposium on Computer and Information Sciences, ISCIS* (Vol. 12, pp. 317-324).

Lewenberg, Y., Sompolinsky, Y., & Zohar, A. (2015). Inclusive blockchain protocols with bitcoin details. *International Conference on Financial Cryptography and Data Security*, 528–547.

Macharla, M. (2018). *Commonly used sensors in the Internet of things (IoT) devices and their application*. Retrieved June 15, 2019, from https://iot4beginners.com/ commonly-used-sensors-in-the-internet-of-things-iot-devices-and-their-application/

Maes, R. (2013). *Physically unclonable functions: constructions, properties and applications*. Berlin, Heidelberg: Springer Berlin.

Mahmoud, R., Yousuf, T., Aloul, F., & Zualkernan, I. (2016). Internet of things (IoT) security: current status, challenges and prospective measures. *2015 10th International Conference for Internet Technology and Secured Transactions, ICITST 2015*, 336–341.

McClelland, C. (2017). *IoT 101: what is a gateway?* Retrieved July 21, 2019, from https://medium.com/iotforall/iot-101-what-is-a-gateway-be066763b98d

Menn, J. (2012). *Key internet operator verisign hit by hackers*. Retrieved July 20, 2019, from https://www.reuters.com/article/us-hacking-verisign/key-internet-operator-verisign-hit-by-hackers-idUSTRE8110Z820120202

Michael, P. E., & James, H. E. (2015). How smart, connected products are

transforming companies. *Harvard Business Review*, *93*(10), 96–114.

Michaluk, K., Nickinson, P., Ritchie, R., & Rubino, D. (2013). *The future of authentication: biometrics, multi-factor, and co-dependency*. Retrieved July 18, 2019, from https://crackberry.com/talk-mobile/future-authentication-biometrics-multi-factor-and-co-dependency-talk-mobile

Moore, R. (2018). *Firefly/wallet · Firefly hardware wallet project - GitHub*. August 14, 2019, from https://github.com/firefly/wallet/blob/master/README.md

Mughal, M. A., Luo, X., Mahmood, Z., & Ullah, A. (2018). Physical unclonable function based authentication scheme for smart devices in Internet of things. *Proceedings - 2018 IEEE International Conference on Smart Internet of Things, SmartIoT 2018*, (Ic), 160–165.

Musienko, Y. (2019). *What are Blockchain smart contracts (Ethereum) and their benefits*. August 18, 2019, from https://merehead.com/blog/what-blockchain-smart-contracts-benefits/

Naik, N. (2017). Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. *2017 IEEE International Symposium on Systems Engineering, ISSE 2017 - Proceedings*.

Narayanan, A., Bonneau, J., Felten, E., Miller, A., & Goldfeder, S. (2015). Bitcoin and cryptocurrency technologies chapter 8: alternative mining puzzles.

Nolan, S. (2018). *Ethereum clique poa vs pow*. August 25, 2019, from https://medium.com/coinmonks/ethereum-clique-poa-vs-pow-11be52cddde1

Novo, O. (2018). Blockchain meets IoT: an architecture for scalable access management in IoT. *IEEE Internet of Things Journal*, *5*(2), 1184–1195.

O'Leary, N. (2018). *Knolleary/pubsubclient · Arduino Client for MQTT - GitHub*. Retrieved August 14, 2019, from https://github.com/knolleary/pubsubclient/blob/master/README.md

Ometov, A., Bezzateev, S., Mäkitalo, N., Andreev, S., Mikkonen, T., & Koucheryavy, Y. (2018). Multi-factor authentication: a survey. *Cryptography*, *2*(1), 1.

Orlicki, J. (2018). *Programming languages intro · ethereum/wiki Wiki · GitHub*. Retrieved August 19, 2019, from https://github.com/ethereum/wiki/wiki/Programming-languages-intro

Paar, C., & Pelzl, J. (2010). *Understanding cryptography*. Berlin, Heidelberg: Springer.

Pandya, D., Ram, K., Thakkar, S., Madhekar, T., & Thakare, B. S. (2015). An overview of various authentication methods and protocols. *International Journal of Computer Applications*, *131*(9), 25–27.

Pegasys. (2018). *Ethereum explained: Merkle trees, world state, transactions, and more*. Retrieved July 30, 2019, from https://pegasys.tech/ethereum-explained-merkle-trees-world-state-transactions-and-more/

Peyrott, S. (2015). *What is and how does single sign-on authentication work?* Retrieved July 15, 2019, from https://auth0.com/blog/what-is-and-how-does-single-sign-on-work/

Prahalad, B. (2018). *Merkle proofs Explained.* Retrieved July 27, 2019, from https://medium.com/crypto-0-nite/merkle-proofs-explained-6dd429623dc5

Rogaway, P. (2011). Evaluation of some blockcipher modes of operation. *Cryptography Research and Evaluation Committees (CRYPTREC)*.

Rouse, M. (2012). *What is single point of failure (SPOF)?* Retrieved July 19, 2019, from https://searchdatacenter.techtarget.com/definition/Single-point-of-failure-SPOF

Rouse, M. (2014). *What is out-of-band authentication?* Retrieved July 14, 2019, from https://searchsecurity.techtarget.com/definition/out-of-band-authentication

Rouse, M. (2015). *What is single-factor authentication (SFA)?* Retrieved July 24, 2019, from https://searchsecurity.techtarget.com/definition/single-factor-authentication-SFA

Rouse, M. (2019). *What is a digital signature and how does it work?* Retrieved July 24, 2019, from https://searchsecurity.techtarget.com/definition/digital-signature

Roussev, V. (2009). Hashing and data fingerprinting in digital forensics. *IEEE Security and Privacy*, *7*(2), 49–55.

Rusnak, P. (2018). *Trezor/trezor-crypto · GitHub*. Retrieved August 14, 2019, from https://github.com/trezor/trezor-crypto/blob/master/README.md

Salman, T., & Jain, R. (2017). Networking protocols and standards for Internet of things. In *Internet of Things and Data Analytics Handbook* (215–238). New York;Wiley.

Scheidt, E. M., & Domangue, E. (2005). Multiple factor-based user identification and authentication. *U.S. Patent 7,131,009*, *2*(12).

Sebastian Peyrott. (2017). *An introduction to Ethereum and smart contracts: an authentication solution*. Retrieved June 14, 2019, from https://auth0.com/blog/an-introduction-to-ethereum-and-smart-contracts-part-3/

Seibold, S., & Samman, G. (2016). Consensus. immutable agreement for the internet of value. *Kpmg*, *26*, 2001–2001.

Sellin, E. (2017). *What exactly is Turing completeness?*. Retrieved July 27, 2019, from https://medium.com/@evinsellin/what-exactly-is-turing-completeness-a08cc36b26e2

Senta, L. (2018). *Where and how application data is stored in Ethereum?* Retrieved July 27, 2019, from https://www.lsenta.io/posts/storage-and-dapps-on-ethereum-blockchain/

Shorish, J. (2018). Blockchain state machine representation. *SocArXiv,* Jan. 2018.

Sicari, S., Rizzardi, A., Grieco, L. A., & Coen-Porisini, A. (2015). Security, privacy and trust in Internet of things: the road ahead. *Computer Networks*, *76*, 146–164.

Sirota, D. (2017). *How OneLogin was compromised and the lessons for the rest of us*. Retrieved July 15, 2019, from https://www.entrepreneur.com/article/295831

Song, J. (2017). *Blockchain 101 - elliptic curve cryptography*. Retrieved August 3, 2019, from https://eng.paxos.com/blockchain-101-elliptic-curve-cryptography

Spacey, J. (2016). *3 Examples of mutual authentication*. Retrieved June 20, 2019, from https://simplicable.com/new/mutual-authentication

Stallings, W. (2010). Cryptography and network security: principles and practice. In *Data*. New York: Pearson.

Stornetta, W. S., & Haber, S. (1991). How to time-stamp a digital document. *Journal of Cryptology*, *3*(2), 99–111.

Suarez-Albela, M., Fernandez-Carames, T. M., Fraga-Lamas, P., & Castedo, L. (2018). A practical performance comparison of ECC and RSA for resource-constrained IoT devices. *2018 Global Internet of Things Summit, GIoTS 2018*.

Suárez-Albela, M., Fraga-Lamas, P., & Fernández-Caramés, T. M. (2018). A practical evaluation on RSA and ECC-based cipher suites for IoT high-security energy-efficient fog and mist computing devices. *Sensors (Switzerland)*, *18*(11), 0–26.

Thomas, S. A., & Wiley, J. (2000). *SSL and TLS essentials - securing the web*. New York: Wiley.

Tudonu, M. (2018). *A deep dive into the Ethereum Virtual Machine (EVM) - part 2: memory and storage*. Retrieved August 7, 2019, from https://kauri.io/article/766e5d1e1ba240a7976943b659a871fc/v1/a-deep-dive-into-the-ethereum-virtual-machine-(evm)-part-2:-memory-and-storage

Unen, C. (2017). *The end of 2FA via SMS*. Retrieved July 20, 2019, from

https://www.cm.com/blog/key-update-nist-no-longer-deprecates-sms-for-2fa-in-digital-identity-guidelines/

Vincent, J. (2019). *Bitcoin consumes more energy than Switzerland, according to new estimate - The Verge*. Retrieved July 28, 2019, from https://www.theverge.com/2019/7/4/20682109/bitcoin-energy-consumption-annual-calculation-cambridge-index-cbeci-country-comparison

Vorakulpipat, C., Rattanalerdnusorn, E., Thaenkaew, P., & Dang Hai, H. (2018). Recent challenges, trends, and concerns related to IoT security: an evolutionary study. *International Conference on Advanced Communication Technology, ICACT*, *2018-Febru*, 405–410.

Wang, S., Zhu, S., & Zhang, Y. (2018). Blockchain-based mutual authentication security protocol for distributed RFID systems. *Proceedings - IEEE Symposium on Computers and Communications*, 74–77.

Whittaker, Z. (2018). *SAML protocol bug let hackers log in as other users | ZDNet*. Retrieved July 20, 2019, from https://www.zdnet.com/article/saml-protocol-bug-puts-single-sign-on-accounts-at-risk/

Witherspoon, Z. (2017). *How is an Ethereum address generated?* Retrieved June 24, 2019, from https://medium.com/@zanewithspoon/how-is-an-ethereum-address-generated-9f08e6f83f77

Wood, G. (2017). *Ethereum: A secure decentralised generalised transaction ledger. eip-150 revision*. Retrieved June 27, 2019, from https://ethereum.github.io/yellowpaper/paper.pdf

Wu, L., Du, X., Wang, W., & Lin, B. (2018). An out-of-band authentication scheme for Internet of things using blockchain technology. *2018 International Conference on Computing, Networking and Communications, ICNC 2018*, 769–773.

Yaga, D., Mell, P., Roby, N., & Scarfone, K. (2018). Blockchain technology overview (nistir-8202). *National Institute of Standards and Technology*, 1–57.

Yang, Z., Yue, Y., Yang, Y., Peng, Y., Wang, X., & Liu, W. (2011). Study and application on the architecture and key technologies for IoT. *2011 International Conference on Multimedia Technology, ICMT 2011*, 747–751.

Yassin, A. A., Jin, H., Ibrahim, A., & Zou, D. (2012). Anonymous password authentication scheme by using digital signature and fingerprint in cloud computing. *Proceedings - 2nd International Conference on Cloud and Green Computing and 2nd International Conference on Social Computing and Its Applications, CGC/SCA 2012*, 282–289.

Ye, N., Zhu, Y., Wang, R. C., Malekian, R., & Lin, Q. M. (2014). An efficient authentication and access control scheme for perception layer of Internet of things. *Applied Mathematics and Information Sciences*, *8*(4), 1617–1624.

Yerlikaya, O., & Dalkilic, G. (2018). Authentication and authorization mechanism on message queue telemetry transport protocol. *UBMK 2018 - 3rd International Conference on Computer Science and Engineering*, 145–150.

York, L. (2018). *What is hashing?* Retrieved June 5, 2019, from https://medium.com/tech-tales/what-is-hashing-6edba0ebfa67

Zhang, V. (2018). *Web3E Ethereum for embedded devices*. Retrieved August 14, 2019, from https://github.com/AlphaWallet/Web3E

Zhang, Y., & Wen, J. (2017). The IoT electric business model: using blockchain technology for the internet of things. *Peer-to-Peer Networking and Applications*, *10*(4), 983–994.

Zhao, K., & Ge, L. (2013). A survey on the internet of things security. *Proceedings - 9th International Conference on Computational Intelligence and Security, CIS 2013*, 663–667.

Zhong, C. Le, Zhu, Z., & Huang, R. G. (2016). Study on the IoT architecture and gateway technology. *Proceedings - 14th International Symposium on Distributed Computing and Applications for Business, Engineering and Science, DCABES 2015*, 196–199.

APPENDICES

Appendix 1: The Genesis Configuration


```
{
    "config": {
      "chainId": 15,
      "homesteadBlock": 1,
      "eip150Block": 2,
      "eip150Hash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
      "eip155Block": 3,
      "eip158Block": 3,
      "byzantiumBlock": 4,
      "constantinopleBlock": 5,
      "clique": { "period": 3, "epoch": 30000 }
    },
    "nonce": "0x0",
    "timestamp": "0x5d0fe88a",
    "gasLimit": "0x47b760",
    "difficulty": "0x1",
    "mixHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
    "coinbase": "0x0000000000000000000000000000000000000000",
    "alloc": {
      "2125879ff1e7fd2eb41de69729920503ee181b39": {
        "balance":
"0x2000000000000000000000000000000000000000000000000000000000000000"
      },
      "75660ebb9cc19eecc3672070e33bfc5e59c45f1e": {
        "balance":
"0x2000000000000000000000000000000000000000000000000000000000000000"
      },
      "9b9d187633d44c9283e662d8855d96d8165cde25": {
        "balance":
"0x2000000000000000000000000000000000000000000000000000000000000000"
      }
    },
    "number": "0x0",
    "gasUsed": "0x0",
    "parentHash":
"0x0000000000000000000000000000000000000000000000000000000000000000"
  }
```

Appendix 2: The Smart Contract

```
pragma solidity  ^0.5.0;

contract Gateway {

    enum Permission { denied, read, write, readwrite }

    struct AccessFrame {
        bytes32 Topic;
        Permission TopicPermission;
        bool isExists;
    }

    address[]  GatewayAdmins;

    mapping(address => AccessFrame[])  Things;

    event aThingSuccessfullyLoggedIn(address _address,bytes32
_topic,Permission _permission,bytes32 _challenge);

    constructor() public {
        GatewayAdmins.push(msg.sender);
    }

    modifier AdminsOnly() {
        bool isAdmin = false;
        for(uint256 i = 0; i < GatewayAdmins.length; i++) {
            if(msg.sender == GatewayAdmins[i]) {
                    isAdmin = true;
            }
        }
        require(isAdmin);
        _;
    }
    function isAdmin(address _searchAddress) private view returns (int256)
{
        for(uint256 i = 0 ; i < GatewayAdmins.length; i++) {
            if(GatewayAdmins[i] == _searchAddress) {
                return int256(i);
            }
        }
        return -1;
    }
```

```
    function addAdmin(address _adminAddress) public AdminsOnly
returns(bool) {
        require(isAdmin(_adminAddress) == -1);
        GatewayAdmins.push(_adminAddress);
        return true;
    }
    function deleteAdmin(address _adminAddress) public AdminsOnly
returns(bool) {
        require(GatewayAdmins.length > 0,'The System Has No
Administrator');
        int256 positionOfAdmin = isAdmin(_adminAddress);
        require(positionOfAdmin > -1,'Admin Account is not found');
        GatewayAdmins[uint256(positionOfAdmin)] =
GatewayAdmins[GatewayAdmins.length -1];
        GatewayAdmins.length--;
        return true;
    }
    function addOrModifyPermission(address _ThingAddress, bytes32
_Topic,Permission  _Permission)  AdminsOnly public returns(bool ) {
        bool recordExists = false;

        for(uint256 i = 0; i < Things[_ThingAddress].length; i++) {
            if(Things[_ThingAddress][i].isExists
            && Things[_ThingAddress][i].Topic == _Topic) {
                Things[_ThingAddress][i].TopicPermission =
Permission(_Permission);
                recordExists = true;
                break;
            }
        }
        if(!recordExists) {

Things[_ThingAddress].push(AccessFrame(_Topic,Permission(_Permission),true
));
        }
        return true;
    }
    function removePermission(address _ThingAddress) public AdminsOnly
returns(bool) {
        delete Things[_ThingAddress];
        return true;
    }
    function requestAuthorization(bytes32  _Topic) public returns
(Permission) {
        Permission _currentState = Permission.denied;

        for(uint256 i = 0; i < Things[msg.sender].length; i++) {
```

```
            if(Things[msg.sender][i].isExists &&
Things[msg.sender][i].Topic == _Topic) {
                _currentState = Things[msg.sender][i].TopicPermission;
                bytes32 _Challenge =
keccak256(abi.encodePacked(msg.sender,_Topic,block.timestamp));
                emit
aThingSuccessfullyLoggedIn(msg.sender,_Topic,_currentState,_Challenge);
            }
        }
        return _currentState;
    }
}
```

Appendix 3: Simple JSON-RPC Proxy

Main.js:

```
var config = require("./HttpProxyConfig");
var proxyFunctions = require("./HttpProxyMethods");
var http = require("http");


http.createServer(onRequest).listen(config.LocalPort);

var proxyFunctionList = Object.keys(proxyFunctions);

console.log("Registered Methods : " + JSON.stringify(proxyFunctionList));

function onRequest(client_req, client_res) {

    if(client_req.method == "POST") {
         client_res.setHeader('transfer-encoding', '');
          var receivedData = "";

          client_req.on('data',function (partial) {
              receivedData += partial;

              if(receivedData.length > 1e6) {
                receivedData = "";
                client_res.writeHead(413, {'Content-Type':
'text/plain'}).end();
                client_req.connection.destroy();
            }
          });
          client_req.on('end',  function () {
              try {
            var bodyObject = JSON.parse(receivedData);
            console.log(bodyObject);
            if(bodyObject.method &&
proxyFunctionList.indexOf(bodyObject.method) > -1) {

                let jsonMethod = proxyFunctions[bodyObject.method];
                let resultObject = jsonMethod(bodyObject.params);
                client_res.write(resultObject);
                client_res.end();

            }
            else {
                    const options = {
```

```
                        hostname: config.UpstreamUrl,
                        port: config.UpstreamPort,
                        path: client_req.url,
                        method: client_req.method,
                        headers: client_req.headers

                    }
                    const proxyRequest = http.request(options,
(proxy_response) => {
                        client_res.writeHead(proxy_response.statusCode,
proxy_response.headers);
                        var responseData = "";
                        proxy_response.on('data', (d) => {

                            responseData += d;
                        });
                        proxy_response.on('end', (d) => {

                          client_res.write(responseData);
                         client_res.end();
                        });
                    });
                    proxyRequest.write(receivedData);

                    proxyRequest.end();

                }

                 }
                 catch(e) {
                     client_res.write("Invalid Data !!!\n");
                     client_res.write(e.message);
                     client_res.end();
                 }
             });
    }

    else {
         client_res.write('Invalid Call\n');
         client_res.end();
    }
}
```

HttpProxyMethods.js:

```
const config = require("./HttpProxyConfig");
const Web3 = require("web3");

const web3 = new Web3(new Web3.providers.HttpProvider("http://"+
config.UpstreamUrl +":" + config.UpstreamPort));

function searchTransactionMethod(params) {
    var resultArr = [];

    for(var i = 0; i < params.length; i++ ) {
        queryObject = params[i];

        if(!queryObject.endBlockNumber)
            queryObject.endBlockNumber = web3.eth.blockNumber;
        if(!queryObject.count)
            queryObject.count = 10;

        if(!queryObject.startBlockNumber && queryObject.fromTransaction) {
            var searchTran =
web3.eth.getTransaction(queryObject.fromTransaction);
            queryObject.startBlockNumber = searchTran.blockNumber;
            if(!queryObject.startBlockNumber) {                    continue;
            }
        }

        for(var i = queryObject.startBlockNumber; i <=
queryObject.endBlockNumber;i++) {
            var pickedBlock = web3.eth.getBlock(i,true);
            if(!pickedBlock)
                continue;
            pickedBlock.transactions.forEach((e) => {
                if(queryObject.count == 0) {
                    return JSON.stringify(resultArr);
                }
                if((queryObject.from.toLowerCase() == e.from.toLowerCase()
|| queryObject.from.toLowerCase() == "*") && (queryObject.to == "*" ||
queryObject.to.toLowerCase() == e.to.toLowerCase())) {
                    resultArr.push(e.hash);
                    queryObject.count--;
                }
            });
        }
    }
    return JSON.stringify({ "result" : resultArr });
```

```
}


var methods = {
    ext_searchTransaction : searchTransactionMethod
}

module.exports = methods;
```

Appendix 4: Sample Transaction Object Belongs To Device Registration

Transaction Object Property	Transaction Object Property Values
Status	0x1 Transaction mined, and execution succeed
blockNumber	267187
Value	0 wei
transaction hash	0x0a73b4843d0d811611da892a2fe5ae1ef37eb98e7c70ba2f69738 7604b156473
From	0xdd04e99ee8ca027f95275c8310669d7770e275d4
To	Gateway.addOrModifyPermission(address,bytes32,uint8) 0xfb595e72eed272b4aebb023c31ba0838cef01797
encoded input	"0x3f226b62000000000000000000000000540b21be69a7af34002 09d61fc3e5d936c7e54d25465737453656e736f7200000000000000 0000000000000000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000002"
decoded input	{ "address _ThingAddress": "0x540b21bE69A7AF3400209D61fc3E5D936C7e54D2", "bytes32 _Topic": "0x5465737453656e736f72" (TestSensor) , "uint8 _Permission": 2 }

Appendix 5: Login Request Transaction Object

Transaction Object Property	Transaction Object Property Values
status	0x1 Transaction mined, and execution succeed
blockNumber	291070
nonce	234
value	0 wei
transaction hash	“0xfe5ae600b2f12ba4d22a9ed4d2a3bace9d8cbeb5bb580f5d9fbbc361effabf9f”
from	"0x540b21be69a7af3400209d61fc3e5d936c7e54d2"
to	"0xfb595e72eed272b4aebb023c31ba0838cef01797"
encoded input	"0x428b28a85465737453656e736f72000000000000000000000000000000000000000000000000000000"
decoded input	{ "bytes32 _Topic": "0x5465737453656e736f72" } (TestSensor)
decoded output	{ "0": "uint8: Permission 2" }
r	“0x876ae0d4416c0fe0aaa275e0d189643f59d81460cc3f7272762ec6bb54c6decc”
s	“0x5cf30f290c785ebdc6ca04d8b6823dbaa8dd095430861f76ace646c47086a728”
v	“0x1b”

Appendix 6: Successful Login Response Transaction Object

Transaction Object Property	Transaction Object Property Values
status	0x1 Transaction mined, and execution succeed
blockNumber	1259847
nonce	305
value	0 wei
transaction hash	"0x52f00ea569d9190c3adb579341f68a2fb13f1cca9de5ad3ebaa358e2a3e6b857"
from	"0xfb595e72eed272b4aebb023c31ba0838cef01797"
to	"0x540b21be69a7af3400209d61fc3e5d936c7e54d2"
encoded input	"0x65323464393934373166303966356130356431656464653832613866353364366161616630366337643033656661353263863613538393331333963306535393538333865646165626239323761626335633553434643664373935333032373330356462366265306336663613963316134626661653231643236646636393934"

decoded input	{
	Token:"0x34b9c869d5bb4a24f434c04856b0d80fdcc582f406caeac3571744b9d1943623"
	Broker:"192.168.2.1"
	IV: "0x3dd47fcffa64ccc69752824e"
	}

decoded output	-
r	"0x59885d969d42ef161ab2e49f5d342391fbe55af44aaeef977aa39ca56b1c1748"
s	"0x68f7c3a2efb826f353b6e2733fbbcda2198121904edfaafc0b50bec13197ea5c"
v	"0x1c"