Ατομική Διπλωματική Εργασία

BLOCKCHAIN AND SMART CONTRACTS IN THE INTERNET OF THINGS (IOT)

Άγγελος Γεωργίου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2023

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Blockchain and Smart Contracts in the Internet of Things (IoT)

Άγγελος Γεωργίου

Επιβλέπων Καθηγητής

Δρ. Βάσος Βασιλείου

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2023

Acknowledgements

I would like to express sincere thanks to all the people who have assisted, each in their own way, to the completion of this dissertation.

I thank first my supervising professor Dr. Vasos Vasiliou, who guided me through the proper articles and papers in understanding the various concepts involved in Blockchain technology and the Internet of Things (IoT), identifying the problem and conceptualizing a solution.

I thank next the supervising assistant Dr. Iacovos Ioannou, who advised on best practices regarding the implementation of the proposed solution after it was done and on including key elements in this report, for the better framing of the concept.

I also thank my friend Stelios, who helped me in several parts of the implementation of the proposed solution, regarding some of the technologies used that I wasn’t quite familiar with.

Last but not least, I thank my friends and family who supported me in a more general way in the everyday life when I worked on this dissertation.

Abstract

Internet of Things is vastly spreading as an industry, at a very rapid pace, transitioning to smart our homes, work environments or even commercial infrastructures. Although, by growing so fast, the networks face an increasing appearance of new challenges, as the communications become more as well. Between these challenges is the effective system management, while guaranteeing security for the devices' identities and interactions. The conventional methods that have been used so far by the broad mass of the industry, turn out to encounter some serious vulnerabilities, such as centralized servers that cause bottlenecks or even single point of failures, which are usually subject to exploitation by intruders.

The technology of Blockchain, is promising new decentralized platforms that can provide safety and independency. As it follows a peer-to-peer architecture, blockchain introduces the term of the distributed ledger, where every detail regarding any transaction that has ever taken place in the network is kept. In blockchain, consensus mechanisms are implemented too, according to which, some special nodes serve the network by providing consensus on its state at any moment, by following specific algorithms that usually involve computations and demand of high processing power.

In this thesis, research is conducted to check whether there is a possibility of integrating the Blockchain technology into the access management of an IoT network of constrained devices, aiming to gain from the benefits that it goes along with. Within this effort, a proof of concept was invented and conceptualized before its implementation was executed. The developed system performs a small-scale access management for constrained IoT devices on the blockchain of Ethereum. The intended scope was a worldwide range of its application. System performance, specifically response time of its functions, seemed to be dependent by the number of the clients, but not at a great extent as the relation was linear. The obtained results reflect only up to a point the system's deployment to the public Ethereum network, such in real life applications, while the tests were conducted on a private Ethereum blockchain that ran locally where no consensus mechanisms were implemented. However, many other factors remained stable and thus this setup can be considered as representative. The acting nodes of the system were simulated by different Ethereum accounts, provided by the environment used for the development, so the examination could take place. Both the limitations and the benefits that were gained from the integration, have been described in detail.

Contents

Chapter 1

Introduction

1.1 Blockchain and potential in IoT

The emergence of blockchain technology has revolutionized the way we think about decentralization, security, and trust in a digital world. Satoshi Nakamoto's invention of Bitcoin in 2009 introduced the concept of blockchain[9], which enabled secure and immutable transactions without the need for intermediaries. Since then, blockchain technology has evolved and expanded into various domains, including healthcare, finance, and supply chain management.

However, the potential of blockchain technology extends beyond cryptocurrencies. It has enabled the creation of decentralized platforms, such as Ethereum and Stellar, that allow the execution of code and the development of decentralized software programs. These platforms can be used to build decentralized applications (dApps) that can potentially solve some of the existing problems in the centralized systems.

In this thesis, we aim to explore the potential of using Ethereum blockchain to power constrained devices in the Internet of Things (IoT) ecosystem. The IoT has the potential to transform various industries by enabling the interconnectivity of devices and machines, but it also poses significant challenges, such as security and privacy concerns, data ownership, and interoperability. We propose to leverage blockchain technology to address some of these challenges by implementing a proof-of-concept access management system that can operate globally.

1.2 Motivation

The current state of IoT systems heavily relies on centralized or brokered paradigms that demand substantial computational and storage capabilities. Unfortunately, these centralized approaches come at a high cost and struggle to scale efficiently to accommodate the ever-increasing number of IoT devices. Complicating matters further, there is a lack of a unified platform that facilitates seamless communication between devices while ensuring compatibility across different manufacturers' offerings.

The conventional client-server model falls short when applied to the intricate IoT ecosystem, which necessitates a decentralized communication approach capable of mitigating the expenses associated with server infrastructure and upkeep. Enter blockchain technology—a promising solution to this predicament. By harnessing a peer-to-peer decentralized communication approach, blockchain enables the sharing of processing power and storage requirements among a vast network of IoT devices without necessitating additional resource allocation.

Employing blockchain as an underlying platform for IoT yields several noteworthy advantages. Firstly, it establishes a secure, immutable, and decentralized ledger, instilling transparency and trust among all participants within the network. Secondly, it fosters interoperability between devices and services offered by various manufacturers, thereby fostering innovation and healthy competition within the IoT landscape.

1.3 Objectives

Currently, in the management of Internet of Things (IoT) devices, there is a reliance on centralized Authentication, Authorization, and Accounting (AAA) servers that handle user requests for accessing various resources of constrained devices. However, this approach falls short in meeting the global interoperability needs of the IoT, especially considering that certain devices may be part of separate management communities during their lifecycle.

The primary objective of this thesis is to explore a more autonomous and shared approach to identity and access management for IoT devices, eliminating the dependence on a single entity for securely storing all data. Our study delves into the distinctions between conventional device management processes and the utilization of blockchain as a database, seeking to shed light on the potential benefits and drawbacks of this alternative approach.

This thesis aims at answering the following research questions:

1. Can we develop a blockchain-based IoT device management system to operate at a worldwide scale?
2. How does blockchain technology benefit IoT?

Chapter 2

Literature Review

There are many other related papers, that have researched this particular field in a similar way; In [13] various access control technologies are examined. Along the examination, modern access control solutions for IoT networks are discussed, both for the industrial and for the research sector. But the paper doesn’t deal with the work already done by standardization organizations in the field of access management for IoT networks. Study [16] conducts short review of conventional architectures that are currently adopted in the management of IoT. Nevertheless, no any of the included proposals follows any strong standard.

This thesis provides an alternative blockchain-based IoT network management system, following a custom architecture. The work conducted within this thesis differs in several aspects from other similar works like [16] and [14] that deal with IoT management frameworks using blockchain. The developed system, was implemented by specifying policy in only one smart contract using only one blockchain network. Unlikely, [14] promoted the creation of many smart contracts, one for each resource-requester pair. Additionally, [14] doesn’t use smart contracts at all and just proposes a new architecture that is based on the use of blockchain’s composite layer between the IoT devices and the industrial applications. Last but not least, both [16] and [14] put the IoT devices into blockchain, while the architecture that was invented within this research keeps them out and offers communication through intermediary entities, called Managers. Although conceptually it sees the devices as nodes that operate internally in the network, technically they don’t share the distributed ledger to perform the transactions between each other.

Chapter 3

Methodology

This chapter explains the methodology followed by this thesis in addressing the problem described in Introduction along with the reasons that this methodology was preferred over others. The chosen methodology is Design Science Research (DSR).

3.1 Design Science Research

As described by [3], DSR indicates the following steps:

1. Problem identification and motivation: Definition of the research problem and justification of the value of the proposed solution. The justification has to aim the motivation of the audience to appreciate researcher's understanding of the problem. The researcher should be able to know the state of the problem and the importance of its solution.
2. Define the objectives for a solution: The objectives of a solution can occur from the problem's definition and knowledge and have to be feasible. They may be quantitative, e.g., how a desirable solution would be better than the conventional one, or qualitative, e.g., how the produced artifact is expected to provide solutions to challenges that haven't been addressed yet. Any objective should be inferred by problem's specification logic.
3. Design and development: Implementation of an artifact as an object of any design where research contribution is embedded. At this point the artifact's functionality and architecture should be determined, before its creation.
4. Demonstration: One or more instances of the problem should be solved. This can be done through experiments, simulations, case studies, proofs, or other appropriate activity.
5. Evaluation: A measurement of whether the artifact actually provides a solution to the researched problem. A comparison usually takes place between the objectives of the

proposed solution and the actual results of the artifact in context. There are many forms that the evaluation could take, and this has to do with the nature of the problem and the artifact. A rebound to step 3 may be needed to improve effectiveness or this extra activity may be left for future work of other projects.

6. Communication: Publishment of all the value that flows out of the research, regarding every aspect of the problem and the designed artifacts, to all the relevant stakeholders.

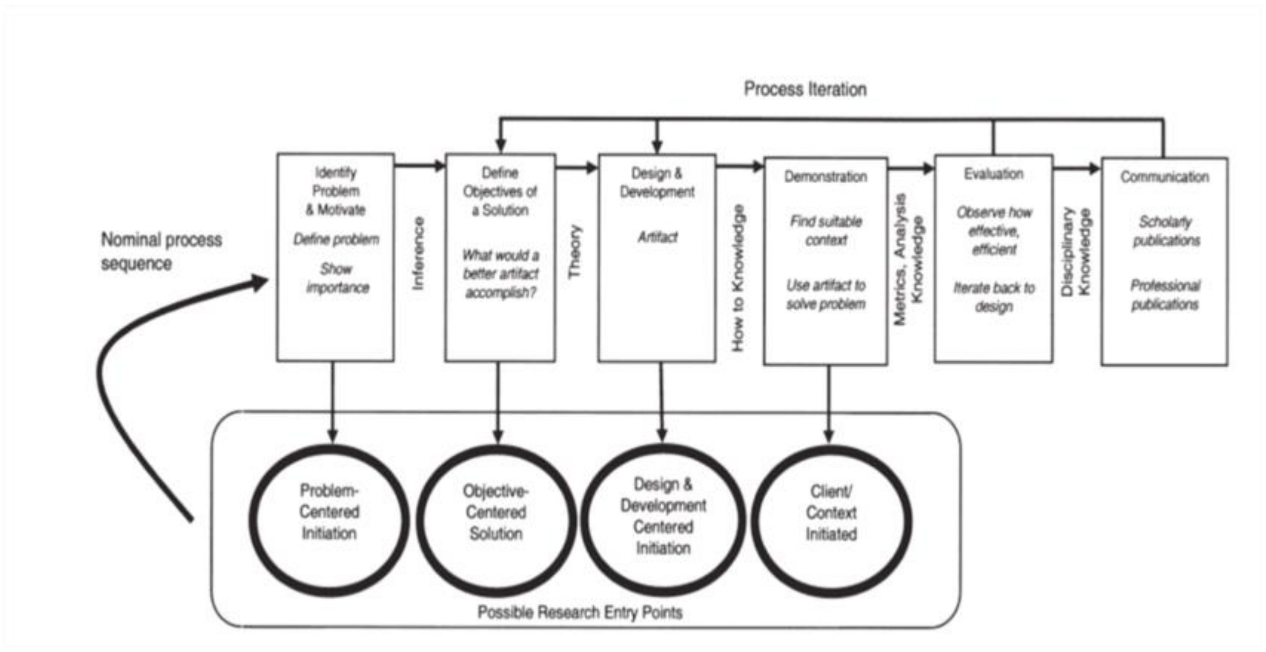

Process Iteration
Nominal process sequence
Identify Problem & Motivate
Define problem
Show importance
Inference
Define Objectives of a Solution
What would a better artifact accomplish?
Theory
Design & Development
Artifact
How to Knowledge
Demonstration
Find suitable context
Use artifact to solve problem
Metrics, Analysis Knowledge
Evaluation
Observe how effective, efficient
Iterate back to design
Disciplinary Knowledge
Communication
Scholarly publications
Professional publications
Problem-Centered Initiation
Objective-Centered Solution
Design & Development Centered Initiation
Client/ Context Initiated
Possible Research Entry Points

Figure 3.1: DSR Process Model[3]

3.2 Advantage

Firstly, DSR is a problem-centric methodology with which real world problems can be solved through design and innovative artifacts. By following DSR’s steps we are explicitly focusing on addressing the problem of blockchain-based IoT device management, ensuring that our research and implementation directly contribute to solving the identified challenges.

Secondly, it is an iterative and incremental approach, allowing us to refine and improve our solution over time. For such complex problems like IoT device management, where there are multiple iterations of design, implementation, and evaluation in order to achieve an optimal solution, this is particularly beneficial. We are let to learn from each iteration and incorporate feedback and insights into subsequent iterations.

We can additionally say that DSR’s structure is based on a design-oriented mindset that emphasizes on the creation of artifacts as means to generate new knowledge and solve problems. This enables to us the design and implementation of a prototype system that demonstrates feasibility and effectiveness on the proposed solution and so the validation of the use of blockchain for IoT management on a global scale.

Furtherly, this methodology has also some practical relevance and application in real-world settings. With its use we don't just expect the development of a theoretical understanding of the problem but also the creation of a practical solution, feasible to implement and evaluate. We are guaranteed that our research contributes to the advancement of knowledge while also provides tangible benefits for practitioners in IoT device management.

Chapter 4

Blockchain

4.1 Introduction to Blockchain Technology and Its Applications

Blockchain technology has garnered much attention since its introduction in 2009 with the launch of Bitcoin, a decentralized digital currency. The technology has come a long way since then, and its potential uses are being explored by various industries, including finance, healthcare, supply chain management, and more recently, the Internet of Things (IoT). In this section, we will provide an overview of blockchain technology, how it works, and its potential applications.

Blockchain is a decentralized, immutable, and transparent ledger that records all transactions in a distributed network. It uses cryptographic techniques, specifically the cryptographic hash function SHA256, to secure transactions and maintain the integrity of the network. The blockchain consists of a series of blocks, each containing a set of transactions, and each block is connected to the previous block, forming a chain of blocks. This structure ensures that once a block is added to the chain, it cannot be altered or deleted without invalidating the entire chain. The system supports a feature called Smart Contract, ensuring that each party is in compliance with the conditions.

One of the most significant advantages of blockchain technology is its decentralization. Instead of relying on a central authority, such as a bank or government, to validate transactions, blockchain uses a network of nodes to verify and validate transactions. This means that transactions can be processed faster, and there is no need for intermediaries, thereby reducing costs and increasing transparency.

Depending on the level of access to which users are involved in the Network, Blockchains have been categorised as three main types. [4] categorizes it into public blockchains, private blockchains and consortium blockchains. Public Blockchains are those in which anyone can participate and read; consortium or permissioned blockchains are the ones that selected their own miners ahead of time, resulting in part decentralisation. Private blockchains are those that can be read by anyone but written to only by private parties such as organisations. The main intention of this thesis is to ensure that the wider system has a mix of Consortium and Public Blockchains. However, the smaller proof of concept used by this thesis will also use a privately funded Blockchain in order to test it.

Blockchain technology has various potential applications. In the financial sector, where it is used to develop decentralised digital currencies such as Bitcoin and Ethereum, it is one of the most popular uses. These currencies have been popular because of their decentralised nature and they are not subject to inflation or government intervention.

The management of supply chains has also been identified as one area where the use of blockchain technology is being investigated. Companies will benefit from using blockchain technology in real time to monitor the movement of goods between manufacturer and user, making it more transparent, reducing costs and improving efficiency.

Blockchain technologies are also being explored by the healthcare sector. For healthcare providers, the use of blockchain will enable them to protect their patient's records and ensure that patients are able to manage access to medical data in order to improve privacy and security.

One of the most promising applications of blockchain technology is in the Internet of Things (IoT). The IoT is a network of interconnected devices that can communicate with each other and exchange data. By using blockchain technology, devices in the IoT network can securely communicate and exchange data without the need for intermediaries, reducing costs and increasing efficiency and scalability.

In the following subchapters we get into details of the operation of blockchain technology and its real world implementations.

4.2 Architecture

Blockchain technology is a distributed database system that provides a secure and transparent platform for recording and verifying transactions. The architecture of blockchain is based on a decentralized, peer-to-peer network known as P2P (on the top of the network layer), where all nodes work together to maintain the integrity of the transaction ledger.

The blockchain network is composed of nodes, which are essentially individual computers that are connected to each other over the Internet. These nodes are meant to reach a consensus on the state to have some copy of the blockchain available at all times, meaning that there is no central authority with responsibility for managing the network.

The transaction ledger, or blockchain, is a continuously growing list of records, called blocks, which are linked and secured using cryptography. Each block contains a cryptographic hash of the previous block, along with a timestamp and a record of new transactions. By linking blocks in this way, the blockchain creates an immutable, tamper-proof record of all transactions on the network. This is actually the fundamental element of the blockchain.

To add a new block to the blockchain, the network of nodes must reach consensus. It means the new block has to be agreed by a majority of nodes that it is valid and must be added to the chain. This process is called 'mining', and it entails dealing with a complicated mathematical problem in which considerable computing power is required. When it has been mined, the block is distributed to all nodes and each of them will verify its validity before they add it to their copy of the blockchain.

The decentralized and distributed nature of blockchain technology provides several benefits, including increased security, transparency, and resilience. Because there is no central point of control, the network is much harder to hack or attack. Additionally, because all nodes have a copy of the blockchain, it is extremely difficult to tamper with or alter the records.

4.2.1 Network Architecture

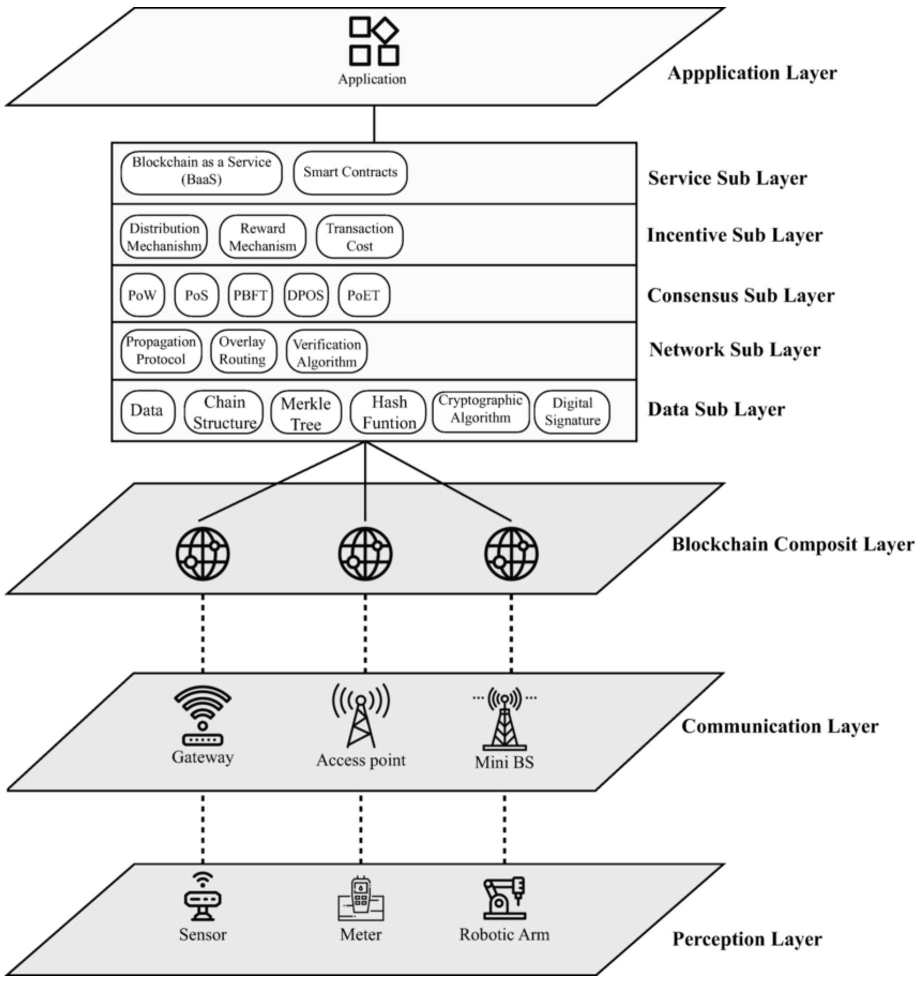


Figure 4.1: Blockchain Network Architecture[15]

4.2.2 Nodes

In a blockchain network, nodes are the individual computers or devices that participate in the network by communicating with one another to maintain the integrity and security of the system. Nodes can be run by anyone who wants to participate in the network, and they can perform a variety of functions depending on their configuration and the role they play in the network.

One of the most common roles that nodes play in a blockchain network is that of miners. The miners are in charge of adding blocks to the blockchain using a process known as proof of work. For the purpose of solving complicated mathematical equations, this requires a considerable amount of energy and computing power.

However, not all nodes in a blockchain network are miners. In fact, there are many different types of nodes, each with their own specific functions and responsibilities. Some nodes are responsible for routing transactions between different parts of the network, while others are responsible for storing the blockchain data and keeping it up-to-date.

There are three main types of nodes in a blockchain network: full nodes, SPV (simple payment verification) nodes, and web clients. Full nodes are the most powerful and secure type of node, as they download and store a complete copy of the blockchain on their computer. This allows them to verify transactions and blocks independently and ensure that the network is functioning properly.

SPV nodes, on the other hand, do not download the entire blockchain. Instead, they only download a small part of the blockchain, such as the block headers, which allows them to verify transactions without using as much computational power. This makes them ideal for mobile devices or other low-power devices.

Finally, web clients are nodes that connect to the blockchain network through a web interface or API. These nodes do not download the blockchain or perform any mining or validation functions, but they can still interact with the network to send and receive transactions.

The node will begin to search for additional nodes that are connected to a specific TCP port as soon as it has entered into the Blockchain Network. It shall thus be capable of communicating and sharing information about transactions, blocks or additional important data with others in the network. Nodes help to ensure the safety, decentralisedness and resilience of the blockchain network through their cooperation.

4.2.3 Transactions

In blockchain, transactions are the basic building blocks of the network. A transaction refers to any movement of value or information that is recorded on the blockchain. These transactions can include sending and receiving cryptocurrency, storing data on the blockchain, and executing smart contracts. They are actually stored data structures, not encrypted and typically linked to previous transactions, thus forming a chain.

Technically, a transaction is a digital message that is broadcast to the network and contains information about the sender, recipient, and the amount or type of asset being transferred. When a transaction is created, it is signed with the private key of the sender to ensure its authenticity and to prevent any unauthorized changes to the transaction.

Once a transaction is set up, it will be broadcast to the network and detected by nodes or miners who are validating transactions and confirming them. Verification of the sender's financial ability to make a transaction, verification of digital signatures and ensuring that transactions comply with the rules of the blockchain protocol shall be part of the validation process.

Once validated, the transaction is added to a pool of transactions which have not been confirmed yet and are expected to be included in this block. The transactions in this pool are competing with each other to join a block that will be added to the blockchain. The miners, who are responsible for the creation of new blocks and to integrate them into the blockchain, choose their transactions from this pool based on transaction fees and priority.

Once a transaction is included in a block, it is considered confirmed and becomes a permanent part of the blockchain. The block containing the transaction is broadcast to the network and is added to the blockchain by other nodes or miners who validate and confirm the block.

In Table 4.1, we see the typical information included in a blockchain transaction. It's worth noting that different blockchains may have slightly different transaction formats and fields.

Size (Bytes)	**Field Name**	**Description**
4	Version	The version number of the transaction format.
1-9	Input Count	The number of transaction inputs.
Variable	Inputs	The inputs of the transaction. Each input includes the previous transaction output being spent and a script that unlocks it.
1-9	Output Count	The number of transaction outputs.
Variable	Outputs	The outputs of the transaction. Each output includes an amount of cryptocurrency and a locking script that specifies the conditions under which the funds can be spent.
4	Locktime	The block height or timestamp after which the transaction can be included in a block.
4	Witness Size	The size of the witness data, used for segregated witness transactions.
Variable	Witness Data	The witness data, used for segregated witness transactions.
4	Transaction Size	The size of the entire transaction, including all fields.

Table 4.1: Information included in a blockchain transaction

4.2.4 Blocks

In blockchain, a block refers to a collection of verified and validated transactions that are added to the blockchain ledger. These blocks serve as a permanent and unalterable record of all the transactions that have taken place on the blockchain network.

Each block is linked to the previous block through the cryptographic hash function SHA256, creating a chain of blocks, hence the name "blockchain". This makes it extremely difficult for any party to alter any transaction that has been recorded on the blockchain, as it would require changing the hashes of all subsequent blocks as well. The linking continues all the way up to the first block in the blockchain, called “genesis” block, usually hardcoded.

In Figure 4.2, we see the genesis block marked dark blue. The current active blockchain consists of the light blue blocks along with the genesis block. The blocks marked in grey are called “stale” blocks and are actually not part of the blockchain from a point and on. When multiple nodes or miners in a blockchain network produce valid blocks at the same time, the network can temporarily split into different chains. This situation is called a fork. Eventually, only one of the chains will become the main blockchain, while the other blocks will be considered stale blocks.

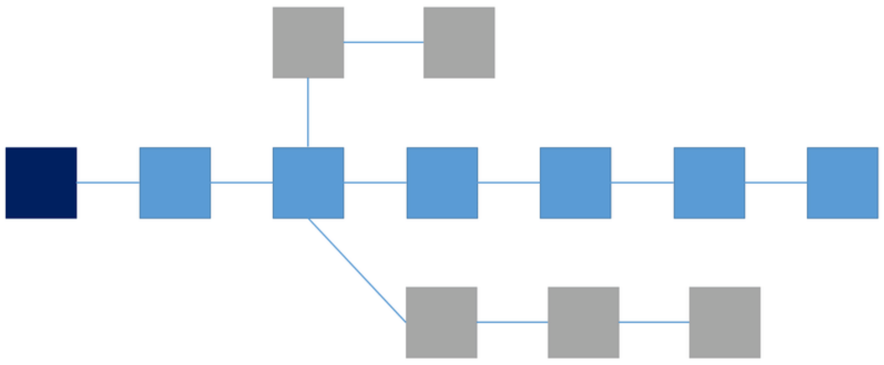


Figure 4.2: Blocks in the blockchain[2]

There are several important pieces of information in blocks, such as transaction data, timestamp and a unique cryptographic hash which is used to identify the block. In addition, a nonce that is an arbitrary value used during the extraction process to generate valid block hash can be present in blocks. In Table 4.2 we see a detailed structure of a block. In Table 4.3 we see furtherly the information that is included in a block header.

Size (bytes)	Field	Description
Variable	Block Size	A 32-bit integer that represents the size of the block in bytes. This field is included to ensure that blocks do not exceed the maximum block size allowed by the network.
80	Block Header	Consists of several fields, as shown in Table 2.3.
Variable	Transactions	A variable-length list of all the transactions included in the block. Each transaction includes information about the sender, recipient, amount, and any fees paid to the miner.
1-9	Transaction Counter	Number of transactions included in a block.

Table 4.2: Structure of a block

Block sizes can change according to the particular network in question, but most blockchains have a minimum block size so that they do not get overwhelmed by massive blocks. A maximum block size of 1 MB can be found in bitcoin.

Size (Bytes)	Field Name	Description
4	Version	A 32-bit integer that specifies the version of the software that the miner used to create the block.
32	Previous Hash	A 256-bit hash of the previous block's header. This ensures that all blocks are linked together in the correct order, forming the blockchain.
32	Merkle Root	A 256-bit hash of all the transactions included in the block. The Merkle root is used to verify that the transactions included in the block are valid and have not been tampered with.
4	Timestamp	A 32-bit integer that represents the time when the block was created. The timestamp is important for maintaining the integrity of the blockchain, as it ensures that blocks are created in the correct order.
4	Difficulty	A 32-bit integer that represents the difficulty level of mining the block. The difficulty level is adjusted automatically by the network to ensure that blocks are created at a steady rate.
4	Nonce	A 32-bit integer that is randomly generated by the miner in order to create a valid hash for the block. The nonce is used in combination with the other fields in the block to create a unique hash that meets the difficulty level required by the network.

Table 4.3: Information included in a block header

Miners, who are nodes on the blockchain network that compete to add new blocks to the blockchain, play a crucial role in the creation of new blocks. They use powerful computing resources to perform complex mathematical calculations in a process known as mining, which helps to validate transactions and create new blocks. The transactions verification process miners compete for is connected with the timestamp, difficulty and nonce fields.

Once the block has been mined and inserted to the blockchain, it will remain a permanently embedded part of the network where there is no change in its content. This creates a high level of safety and tamper resistance for the blockchain technology.

The validation process of the blocks that leads to the addition of a block to the blockchain takes place in the following sequence. The system shall first verify whether a prior block hash that is currently referenced by an existing block has been maintained. This time stamp shall be checked as the block must have a value which is higher than or equal to that of the previous block and less than 2 hours into the future. The validity of proof-of-work is then checked. The new block is added to the blockchain when the condition of the previous block returns true, compared with the list of transactions.

4.2.5 Merkle Trees

Merkle Trees are a data structure used in blockchain technology to efficiently and securely store and verify large amounts of data.

The purpose of a Merkle Tree is to provide a way to confirm that a large set of data has not been tampered with. The tree is constructed by recursively hashing pairs of transactions until a single root hash is obtained, which represents the entire set of transactions[1]. Each layer of the tree represents a set of hashes from the layer below it, and the root hash is the final result. This creates a hierarchical structure where each node is a hash of its children, leading up to the root hash at the top.

Merkle trees are used to store and verify the integrity of all transactions in a block in the context of the blockchain. Each transaction is hashed and added to the Merkle Tree. The root hash of the Merkle Tree is then included in the block header, which is itself hashed and used as a component in the proof-of-work algorithm. This ensures that all subsequent hashes in the Merkle Tree will need to be recalculated if any change is attempted on a block transaction, thereby making it computationally impossible to manipulate data. In Figure 4.3 we see how Merkle Trees are used in transactions in a block of blockchain.

By facilitating the efficient and safe checking of large sets of data, Merkle Trees are playing a crucial role as part of the safety of Blockchain technology. This allows for a way to ensure that the block's information remains valid and has not been damaged, which is vital if we are to maintain trust in the Blockchain network.

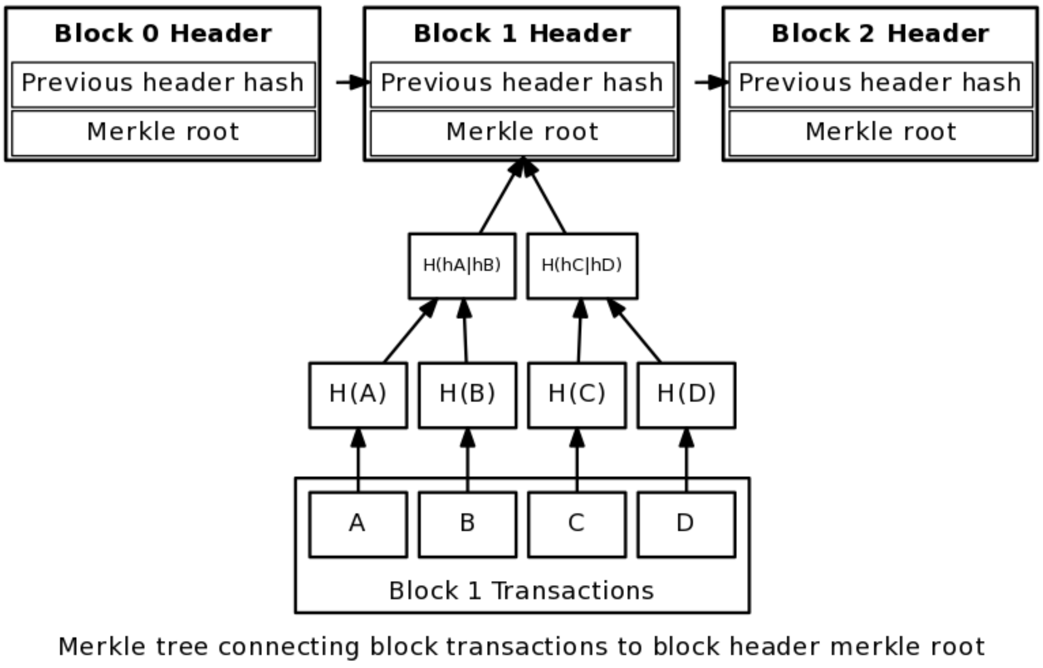


Figure 4.3: Merkle Trees in a block[11]

It is not necessary to have a copy of the whole tree because every Merkle root contains a hash of all transactions in the block.

4.2.6 Mining

Mining in blockchain refers to the process of adding new blocks to the blockchain by solving complex mathematical puzzles through the use of computational power. In essence, miners compete to solve the mathematical problem first, and the winner gets to add the next block to the blockchain and receive a reward in the form of cryptocurrency.

It is a competitive process, with many miners racing to solve the problem and claim the reward. This competition ensures the security and decentralization of the network, as no single entity can control the majority of the mining power.

A block may also have the possibility of being generated by two or more miners simultaneously. This leads to the formation of the “stale” blocks. Each blockchain implementation deals differently with these blocks.

The steps involved in mining a block are as follows:

1. Transaction validation: The first step in the mining process is to validate the transactions that are included in the new block. This involves verifying that each transaction is valid and has not been previously spent.
2. Creating a block header: The miner then creates a block header, which includes the previous block hash, a timestamp, a nonce, and a list of transactions.
3. Finding a nonce: The miner then begins the process of finding a nonce that, when added to the block header, produces a hash value that meets a specific difficulty target. This process involves trial and error, as the miner must repeatedly hash the block header with different nonce values until a valid hash is found.
4. Broadcasting the block: Once a valid hash is found, the miner broadcasts the new block to the network. Other nodes on the network then verify the new block and add it to their copy of the blockchain.
5. Block reward: As a reward for their efforts, miners receive a block reward in the form of new cryptocurrency units. In the case of Bitcoin, for example, the block reward is currently 6.25 BTC per block.

The biggest known example of blockchain mining has been Bitcoin mining. In Bitcoin, miners are competing to solve a mathematical problem in which they attempt to find the hash value that meets certain criteria. The computational power required to run this operation is significant, and the math problem that miners are going to have to solve with growing networks also becomes more difficult. At the moment it is just profitable to mine bitcoins with specially designed mining equipment, like ASICs (application-specific integrated circuits).

By contrast, Ethereum is using a different algorithm for its mining operations called Ethash, which has been designed to be more memory intensive and is aimed at preventing special mining equipment from dominating the network. There are still GPUs that could be used for Ethereum mining, although ASICs have already been developed for Ethash.

There are two types of mining: solo mining and pool mining. Solo mining involves an individual miner using their own resources to mine and validate transactions. Pool mining, on the other hand, involves a group of miners combining their resources to increase their chances of solving the mathematical problem and earning the reward and is more common than solo mining, as it allows miners to share the costs and risks associated with mining.

4.3 Consensus Mechanisms

In distributed systems, consensus mechanisms have a critical role to play and are particularly important in the blockchain ecosystem. In order to guarantee consistency and safety, a distributed system requires that each node agrees on the position of the system at present. A consensus mechanism makes it possible for nodes to agree with one another on the current state of the scheme without having to have a centralised body at their disposal. In a blockchain network, consensus is required to verify and add new blocks to the blockchain, ensuring that all participants have the same copy of the distributed ledger.

Blockchain networks use various consensus mechanisms, each with its own unique set of benefits and drawbacks. These mechanisms are designed to ensure that the network remains secure, decentralized, and resistant to attacks. Some of the most commonly used consensus mechanisms include Proof of Work (PoW), Proof of Stake (PoS), and Delegated Proof of Stake (DPoS). Each of these mechanisms uses different algorithms and incentives to achieve consensus among the network participants.

In the following sections, we will explore these consensus mechanisms in more detail, including their advantages, disadvantages, and how they operate in a blockchain network.

4.3.1 Proof of Work

The PoW algorithm involves a competition between nodes to solve a mathematical problem known as the hash puzzle. The puzzle requires miners to find a nonce, a random number, that when added to the transaction data of a block, produces a hash value that meets a specific set of criteria[5]. The criteria are set by the network difficulty level and are adjusted regularly to maintain a consistent rate of block creation. PoW is really a brute force crack on a SHA256 hash algorithm. This works as a proof that the miner has used some heavy duty computing resources. The exact condition is that the double hash of every block must be less than the difficulty target.

The first miner to solve the hash puzzle and find a valid nonce broadcasts the block to the network, and other miners verify the solution. Once verified, the block is added to the blockchain, and the miner is rewarded with a specific amount of cryptocurrency, which incentivizes miners to participate in the network.

However, the PoW algorithm has several drawbacks, including high energy consumption and the centralization of mining power among a few large mining pools. Another major drawback of PoW is the possibility of a 51% attack. This occurs when a single miner or a group of miners control more than 50% of the network's computing power, enabling them to manipulate the blockchain. In such a scenario, the attacker(s) can potentially reverse transactions, exclude certain transactions from being included in new blocks, or even double-spend coins. This undermines the trust in the network and reduces its reliability. As a result, many blockchain networks have been exploring alternative consensus mechanisms such as Proof-of-Stake (PoS) and Proof-of-Authority (PoA).

PoW continues to be a widely used consensus mechanism, especially on the Bitcoin network, which uses it to verify transactions and insert new blocks into the blockchain despite its limitations.

4.3.2 Proof of Stake

In PoS, validators (sometimes called "forgers" or "minters") are chosen to validate transactions and add new blocks to the blockchain based on the amount of cryptocurrency they hold or "stake" in the network. The more cryptocurrency a validator has, the greater their chances of being selected to create a new block.

Validators are incentivized to act honestly and maintain the integrity of the network, as they risk losing their staked cryptocurrency if they engage in malicious behaviour such as attempting a double-spending attack.

The process of creating a new block in PoS typically involves the following steps:

1. Validators place a certain amount of cryptocurrency into a special wallet known as a "staking wallet".
2. Validators are randomly selected to validate transactions and add new blocks to the blockchain.

3. The selected validator creates a new block, validates transactions, and signs the block with their staked cryptocurrency.
4. Other validators on the network check the block for accuracy and vote to accept or reject it.
5. If the block is accepted, the validator who created it receives a reward in the form of newly minted cryptocurrency.

As anyone with a cryptocurrency can take part in consensus process, PoS will have better energy efficiency and more transparent distribution of rewards than PoW. However, PoS is not without its drawbacks, such as the potential for centralization, if a small number of validators hold a large percentage of cryptocurrencies, and the possibility of "nothing at stake" attacks, if validators have nothing to lose by creating a fork.

Several prominent blockchain networks, including Ethereum, Cardano, and Binance Smart Chain, have adopted or are in the process of transitioning to a PoS consensus mechanism.

4.3.3 Delegated Proof of Stake

DPoS is a modification of the Proof-of-Stake (PoS) consensus algorithm that seeks to increase scalability and reduce the risk of centralization. It combines PoW and PoS characteristics, using a decentralized voting system [8] .

A number of delegates for validating transactions and creating new blocks can be chosen by token holders in the DPoS network. The network is maintained by these delegates and they receive compensation in the form of transactions fees as well as blocks. The more tokens a user holds, the more voting power they have, which enables them to participate in the election of delegates.

DPoS aims to achieve faster transaction speeds and a more democratic governance system than traditional PoS. By electing delegates to perform the validation, the network can avoid the need for every node to validate every transaction, which can significantly increase the network's transaction throughput.

DPoS, however, has come under fire for imposing a degree of centralisation on the blockchain network. Because only a select few nodes are responsible for validating transactions and creating new blocks, these nodes hold significant power over the network's operation. This can lead to possible problems such as collusion and vote-buying among the elected delegates.

4.3.4 Proof of Authority

In PoA, instead of relying on miners to validate transactions and add new blocks to the chain, a set of pre-approved validators (referred to as authorities) are responsible for this process.

For instance, the authorities of PoA are typically designated with their public keys and they're estimated to hold a limited amount of digital currency so as to be authorised to confirm transactions and insert new blocks into the chain. The fact that the authorities are financially incentivised to act in good faith and in the best interest of the network, as any unlawful behaviour could leave them with a decrease in their shareholding.

The process of adding new blocks in a PoA network is similar to that of PoW and PoS networks, where a consensus algorithm is used to determine which authority will validate the next block. However, unlike PoW and PoS, the PoA consensus algorithm does not require complex computations or significant amounts of computational power. Instead, it relies on a round-robin process where each authority takes turns validating transactions and adding new blocks to the chain.

One of the advantages for PoA is that it offers more energy efficiency than PoW, because it does not need a lot of computational power. Moreover, due to the fact that PoA networks are relying on a pre-approved set of validators, which requires more validation and verification steps, they can be faster and more effective as compared with other consensus mechanisms.

As the PoA network is based on only a limited number of pre-approved validators to verify transactions and add new blocks to the chain, one of its biggest disadvantages is that it's more centralized[10]. If too many authorities cooperate or compromise each other, they can put the network at risk of attacks. In addition, questions on decentralization and censorship resistance can arise if a set of approved validators are used.

4.4 Double Spending

Double spending[9] is a potential issue in digital transactions where a user can spend the same digital asset (such as cryptocurrency) more than once before the transaction is confirmed and recorded in the ledger. This can happen because digital assets can be easily replicated, unlike physical assets.

In blockchain technology, double spending is prevented by the decentralized consensus mechanism. When a user initiates a transaction, it is broadcasted to the entire network. The network's nodes then validate the transaction and update their copy of the ledger. Once the transaction is validated and recorded in the ledger, it becomes a permanent part of the blockchain, and the digital asset associated with the transaction is transferred from the sender to the receiver.

If a user tries to spend the same digital asset again, the network's nodes will detect the double spend attempt because the ledger will already contain a record of the original transaction. The nodes will reject the second transaction as invalid, and the sender will not be able to double spend the same digital asset.

Chapter 5

Blockchain Implementations

5.1 Bitcoin

Bitcoin is a decentralized digital currency that enables peer-to-peer transactions without the need for a central authority, such as a bank or government. It was created in 2009 by an anonymous individual or group under the pseudonym Satoshi Nakamoto and has since become the world's most widely-used cryptocurrency.

Unlike traditional currencies, bitcoin is based on a decentralised system of ledgers, blockchain, and maintained by computer networks around the world. The transactions are stored on a blockchain that is safe and confidential, which makes it difficult for any data to be distorted or manipulated.

Bitcoin can be acquired through mining or purchased on exchanges or received as payment for goods and services.

One of the unique features of Bitcoin is its fixed supply, which is limited to 21 million coins. This scarcity, combined with the decentralized nature of the network, has led to volatility in its price, with large swings in value over relatively short periods of time.

Bitcoin has also been associated with controversy, with some critics arguing that it is used for illicit activities such as money laundering and drug trafficking. Proponents of this currency are, however, advocating its potential to be a decentralised, boundaryless currency that could empower individuals and undermine conventional financial systems.

5.1.1 Bitcoin Network

The Bitcoin network is a decentralized peer-to-peer network that enables the transfer of bitcoins between users. It consists of a group of nodes, each running the Bitcoin software, which communicates with each other to maintain a shared ledger of all Bitcoin transactions.

The consensus mechanism of PoW, which serves as a basis for verifying transactions and adding new blocks to the blockchain platform, is employed on the Bitcoin network. In this process, a complicated mathematical problem is to be resolved that demands significant computing power. The first node to solve the problem broadcasts its solution to the rest of the network, and if the other nodes agree that the solution is valid, a new block is added to the blockchain and the node that solved the problem is rewarded with newly minted bitcoins.

Three main categories of nodes are able to be classified in the Bitcoin Network: Full Nodes, Lightweight Nodes and Mining Nodes. Full nodes store a complete copy of the blockchain and validate all new transactions and blocks, while lightweight nodes rely on full nodes to provide them with transaction information. Mining nodes, as mentioned earlier, are responsible for creating new blocks and are typically equipped with specialized hardware (ASICs) to perform the necessary computations. Bitcoin’s block time, meaning the time it needs to be mined, is 10 minutes.

Before adding them to the blockchain, transactions in the Bitcoin network are distributed on the network and validated by nodes. To prevent double spending, each transaction is recorded in a block, which is then added to the blockchain in a sequential and immutable manner. This makes it impossible to delete or modify transactions when they are added to the blockchain.

A system of incentives to promote node involvement and block formation is also included in the Bitcoin network. Fees that users pay to have their transactions processed more quickly and newly minted bitcoins awarded to mining nodes so they can successfully add new blocks to the Blockchain are included in this.

5.2 Ethereum

Ethereum is a blockchain-based decentralized platform that enables developers to build decentralized applications (DApps). Launched in 2015, it has quickly become one of the most

popular blockchain platforms in the world, thanks to its innovative technology and broad community support.
The fact that it can execute smart contracts is one of the key features of Ethereum. Smart contracts are self-executing contracts in which the terms of the contract between the buyer and the seller are written directly into code. A wide variety of processes, from finance transactions to voting systems and supply chain management, can be automated with their use.

Developers can write smart contracts using a programming language called Solidity, which is specifically designed for the Ethereum platform. Solidity allows developers to create complex logic that can be executed on the Ethereum Virtual Machine (EVM), which is the runtime environment for smart contracts on the Ethereum network.

Ethereum also has its own cryptocurrency, called Ether (ETH), which is used to pay for transactions on the network and incentivize miners to validate new transactions and add them to the blockchain. The Ethereum network has a block time of around 15 seconds, which is much faster than Bitcoin's 10-minute block time, making it more suitable for applications that require faster transaction times.

5.2.1 Ethereum Accounts

Ethereum Accounts are objects with a unique 20-byte address as identifier on Ethereum network and state transitions, the data transferred between two accounts. There are two types of accounts: Externally Owned Accounts (EOA) and contract accounts.

Externally owned accounts are controlled by private keys and can be used to send ether and interact with smart contracts. They've got a balance in the ether and they can be created by anyone. The transaction shall be signed with the private key associated to the account, when it is sent by an external owned account.

Contract accounts are created by deploying a smart contract to the Ethereum network. They do not have a private key and are controlled by the rules programmed into the smart contract. They hold code and data and can interact with other contracts or externally owned accounts.

There are two fundamental units of Ethereum: Transactions and messages.

A transaction represents a transfer of Ether from one account to another. It can also contain data and instructions for executing smart contracts. The fields included in a transaction are [19]:

- from – the address of the sender, that will be signing the transaction. This will be an externally-owned account as contract accounts cannot send transactions.
- recipient – the receiving address (if an externally-owned account, the transaction will transfer value. If a contract account, the transaction will execute the contract code)
- signature – the identifier of the sender. This is generated when the sender's private key signs the transaction and confirms the sender has authorized this transaction
- nonce - a sequentially incrementing counter which indicates the transaction number from the account
- value – amount of ETH to transfer from sender to recipient (denominated in WEI, where 1ETH equals 1e+18wei)
- data – optional field to include arbitrary data
- gasLimit – the maximum amount of gas units that can be consumed by the transaction. The EVM specifies the units of gas required by each computational step
- maxPriorityFeePerGas - the maximum price of the consumed gas to be included as a tip to the validator
- maxFeePerGas - the maximum fee per unit of gas willing to be paid for the transaction (inclusive of baseFeePerGas and maxPriorityFeePerGas)

Transactions require a fee and must be included in a validated block. We discuss about Gas in Subchapter 5.2.2, where Smart Contracts are furtherly analysed.

On the other hand, a message is a similar concept to a transaction, but it is used to invoke the code of a smart contract without necessarily involving a transfer of Ether. It can be sent between contracts or from an EOA to a contract. The fields included in a message are similar to those in a transaction, but they do not include the value field.

5.2.2 Smart Contracts

Nick Szabo firstly introduced and defined the term of Smart Contract in 1994, within his effort to create a self-enforcing and perpetual computer software that would replace the standard legal contract.

“A smart contract is a computerized transaction protocol that executes the terms of a contract. The general objectives of smart contract design are to satisfy common contractual conditions (such as payment terms, liens, confidentiality, and even enforcement), minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries. Related economic goals include lowering fraud loss, arbitration and enforcement costs, and other transaction costs[20].”

In the context of Ethereum, a smart contract is a self-executing contract with the terms of the agreement between buyer and seller being directly written into code and stored on the Ethereum blockchain. Smart contracts enable the creation of decentralised applications (DApps) that can be operated independently and transparent in a manner which does not require intermediaries.

Working on a broad range of tasks, from managing digital assets like tokens to executing complicated financial transactions and the creation of autonomous decentralised organisations (DAOs), can be carried out through smart contracts in Ethereum.

When a smart contract is deployed on the Ethereum network, it becomes a permanent part of the blockchain and can be accessed and interacted with by anyone with an internet connection. Transactions that involve smart contracts can be initiated by anyone on the network, and the results of these transactions are recorded on the blockchain, providing a high level of transparency and security.

We can clearly see that in the case of Ethereum, a smart contract is more like an application with instructions that is invoked when a transaction takes places to the corresponding Ethereum account, rather than just an agreement between two sides. The instructions are written on a high level, Turing complete and Object-oriented scripting language (Solidity) that is executed on the Ethereum Virtual Machine (EVM).

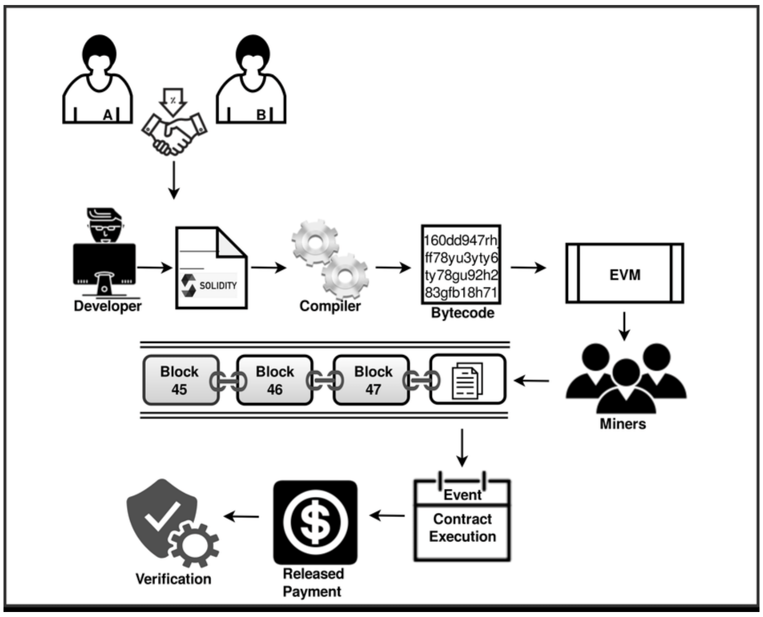


Figure 5.1: Cycle of Smart Contract execution[18]

To measure the computational effort required to execute a transaction or smart contract, Ethereum network uses the unit of Gas. Every operation in the EVM requires a certain amount of gas, which is paid for using ether.

Gas is also a measure for preventing denial-of-service (DoS) attacks on Ethereum network, as the consumed computational resources have to be paid by the users. The amount of gas required for a transaction or contract execution depends on the complexity of the operation being performed. The more complex an operation is, the more gas is needed.

Gas price is the amount of ether a user is willing to pay per unit of gas and is usually set by the developer. Since they can get more ether from the inclusion of these transactions in the blockchain, mutinous miners are prioritising transactions with higher gas prices. Consequently, higher gas prices will generally be established by users who wish to have their transactions completed as soon as possible.

When a transaction runs out of gas during execution, all changed made to the blockchain are rolled back. This ensures that the network remains stable and secure, as it prevents unfinished transactions from being included in the blockchain.

Each operation consumes a fixed amount of gas. Table 5.1 shows the gas amounts consumed by various standard operations.

Operation	**Gas Consumed (Units)**
ADD	3
MUL	5
SUB	3
DIV	5
EXP	10
AND	3
OR	3

Table 5.1: Gas consumed for each operation

Chapter 6

Management in IoT networks

6.1 Protocols and Standards in IoT

Internet of Things as a term was firstly introduced by Kevin Ashton in 1999, while presenting at Procter & Gamble company. He suggested the linking of the then-unheard-of idea of putting wireless sensors on the products to the then-red-hot-topic of the Internet. The IoT refers to a network of physical objects or "things" that are embedded with sensors, software, and other technologies that enable them to connect and exchange data with other devices and systems over the internet. These objects can range from household appliances and wearable devices to industrial machinery and infrastructure. Its goal is to enable a vast network of interconnected devices to communicate with each other and to provide valuable insights and services that can improve efficiency, productivity, and quality of life.

An IoT network's main core of devices consists of Resource Constrained Devices (also known as sensors, smart objects, or smart devices) with limited CPU, memory, and power resources. These devices can't run demanding but essential protocols and implement extra functionalities. Security protocols that are too heavy are delegated to the gateway which acts as an intermediary to support security measures. Given that data retrieval from them is challenging, this thesis will furtherly discuss about CoAP application layer protocol in the following subsections.

6.1.1 Constrained Application Protocol (CoAP)

CoAP is a specialized web transfer protocol designed for constrained devices in the IoT ecosystem. It is designed to use minimal resources, such as memory, processing power, and network bandwidth to facilitate machine-to-machine (M2M) communication in low-power, low-cost devices with limited processing capabilities. Common applications for CoAP include smart energy and smart building automation.

CoAP operates on top of the User Datagram Protocol (UDP) instead of the Transmission Control Protocol (TCP), which makes it lightweight and suitable for the resource constrained devices, using a response/request model similarly to HTTP.

Messages of the protocol contain a code, a method, an identifier, and an optional payload. CoAP supports four request methods: GET, POST, PUT, and DELETE, which allow devices to perform read, write, and delete operations on resources. It also offers features that make it perfect for the use of IoT applications like caching, observation, block transfer and group communication.

6.1.2 Lightweight M2M (LWM2M)

LWM2M is a protocol designed for device management of IoT devices. It was created by the Open Mobile Alliance (OMA) and is used for remote management of sensors, smart meters, and other IoT devices. The protocol is based on and running on the top of CoAP and compatible with any device working with CoAP as its transport protocol.

The LWM2M standard will allow for the communication of IoT devices to management platforms, so that they can be managed remotely over the Internet. This specifies a set of common objects and resources which can be used to configure devices, such as device registration, the update of software or monitoring its status and performance.

Support for device firmware updates via air (FOTA) has been one of the major features of LWM2M. This means that software updates can be forwarded remotely to the device and you don't have to physically access it. This is a key element for IoT devices that may be deployed in remote or difficult to reach locations.

In order to ensure that communications between devices and management platforms are secure and reliable, the LWM2M also contains security elements such as Transport Layer Security (TLS) and message integrity checks.

There are three main parts that form the architecture of LWM2M protocol. A bootstrap server that configures the client before accessing the LWM2M server, a LWM2M server that manages the clients through management commands and a client that is usually a constrained device. Following four logical interfaces are defined between the server and client [17]:

1. Bootstrap: allows LWM2M Bootstrap Server to manage the keying, access control and configuration of a device to enrol with a LWM2M Server.
2. Device Discovery and Registration: allows an LWM2M Client device let the LWM2M Server know its existence and register its capability.
3. Device Management and Service Enablement: allows the LWM2M Server to perform device management and M2M service enablement by sending operations to the Client and to get corresponding responses from the LWM2M Client.
4. Information Reporting: allows the LWM2M Client to report resource information to the LWM2M Server; can be triggered periodically or by events.

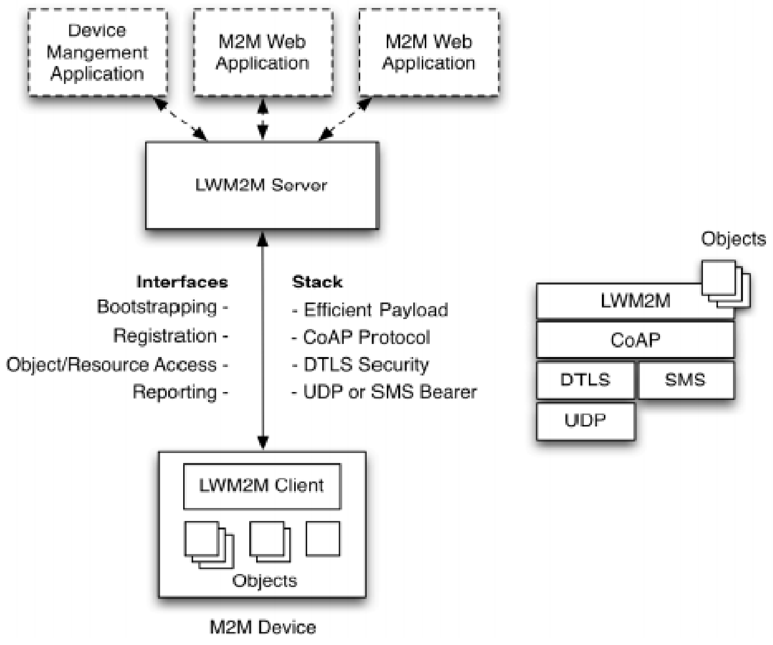


Figure 6.1: LWM2M Architecture[17]

There are 2 well known implementations of LWM2M protocol by Eclipse, both in Java and C, called Leshan and Wakaama accordingly.

6.2 IoT on Blockchains

Blockchain, as an immutable ledger of transactions, can provide fertile ground for the secure co-ordination and communication between billions of IoT devices. By its nature, as described in depth in the previous sections, blockchain can operate in trustless environments and is difficult to manipulate.

This thesis suggests viewing the IoT devices as blockchain nodes. There is no need of the devices to trust each other, while the consensus mechanisms of the blockchain let them so.

Independently of their technical characteristics, any of the IoT devices can interact with others through a common network. If this takes places worldwide, a user centred behaviour on the web could be established in oppose with the existing device centred behaviour. Some constrained IoT devices, with limited resources, may not have the capacity to host a blockchain node and so they may be used as lightweight clients. If they do have the proper capacity, they may be used as representative blockchain nodes for the others. Under this mentality, new hard-to-achieve use cases could be generated where the devices can take the control and management of networks and become autonomous. For example, a smart thermostat could detect when a user has left the house and adjust the temperature accordingly, while a smart lock could automatically lock the doors when the user is away.

Chapter 7

Proof of Concept

In order to prove the successful development and operation of a blockchain-based management system for constrained IoT devices, a small-scale prototype was implemented. Specifications regarding the logic behind the choice of the used technologies and the architecture of the developed system are discussed in more extent in the following subsections. Should be noted that the rationale of the system was influenced by [12].

7.1 Rationale behind technology

In order to create a peer-to-peer distributed system with the demanded characteristics, a blockchain network had to be chosen. Ethereum emerged as the most suitable out of all the available blockchain networks that are broadly offered for development, for various reasons.

First and foremost, Ethereum supports the execution of smart contracts. It is the most widespread platform for smart contract deployment and execution worldwide, with a built-in environment dedicated to these processes. In this way, by creating a dApp, we can more easily implement the device management logic in a decentralized and automated way. It is also remarkable that the Solidity language that was used within the implementation's context for the development of the logic, is very flexible and easy-to-learn.

What's more, Ethereum has a relatively very fast transaction processing, with its block time fluctuating around several seconds, unlike other blockchains' ones that takes minutes e.g., Bitcoin that has 10 minutes for block time. This is particularly important for IoT devices, which require low-latency and real-time responsiveness.

7.2 Architecture

This thesis aims to provide a solution for the management of a very large number of interconnected IoT devices. The management system that was invented within this context is analysed in detail in the current subchapter. A high-level overview of the conceptualized architecture is provided in Figure 7.1 below.

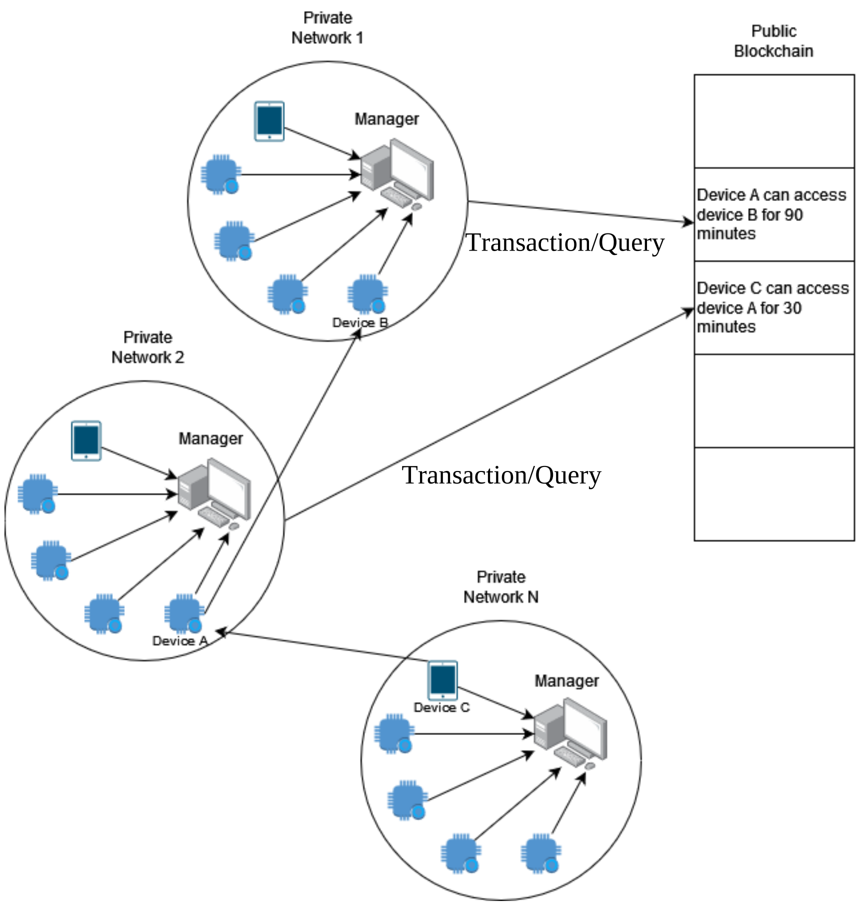


Figure 7.1: Architectural Overview

The IoT devices of the network are constrained and so they do not have the capability to host an Ethereum ledger on their own, due to storage limitations. As illustrated in [12], a solution to this challenge is the set up of some representative devices with greater resource inventories, through which the devices of the network can request queries from the public blockchain. These representative nodes are *Managers* and are connected to that public network, having an administrative role. Each constrained device has to be registered under a Manager's control in order to be able to interact with the network and the access control that they are permitted is specified by the assigned Manager. Managers have to be registered in the network as well. Figure 7.2 illustrates a use case diagram of the system.

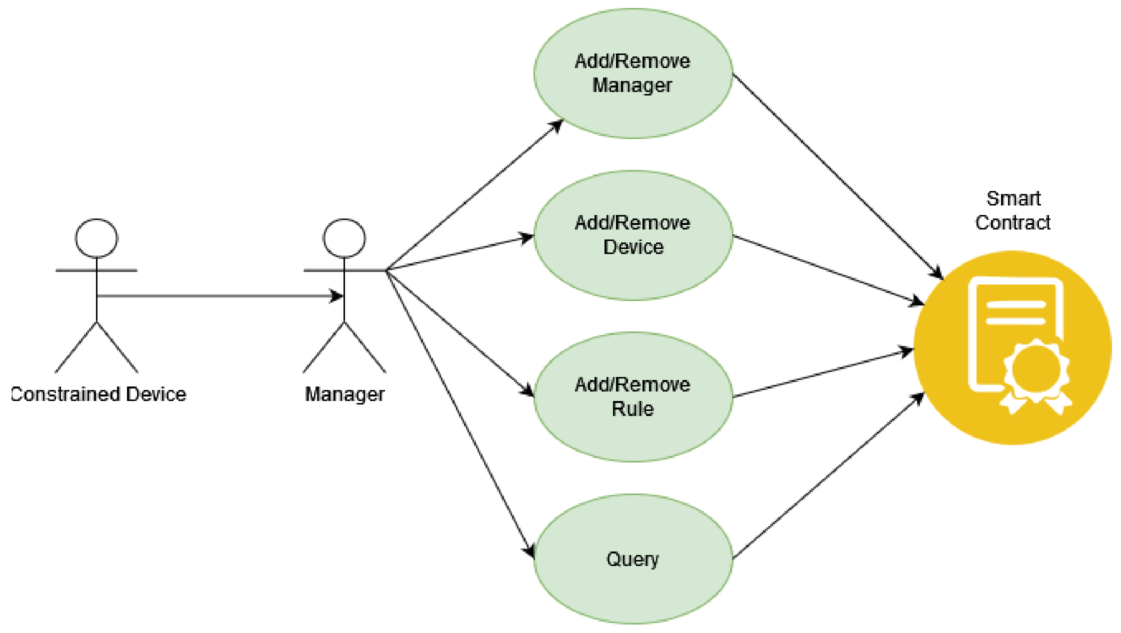
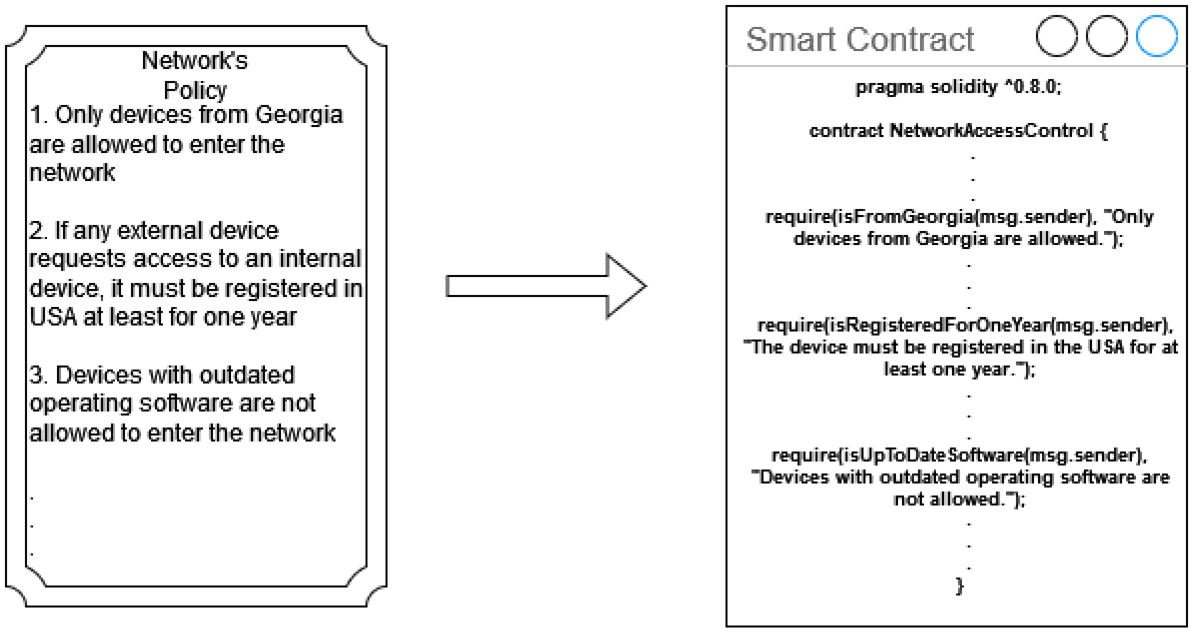


Figure 7.2: Use Case Diagram

It should be noted that Managers do not act as intermediary proxies where managing decisions are taken. They are more like hubs that forward devices' requests to the public blockchain network, where decisions are approved in a decentralized manner by the miners, depending on the governing smart contract's policy. This thesis only deals with only a simple policy within the context of the small-scale prototype and only several management functions are offered. More details are analysed in the relevant subchapter that follows. No misunderstanding should occur, the architecture does not agree to any semi-centralized management structure but to a completely centralized one with full dependency on the decentralized environment of blockchain and smart contracts.

When applied to any more complex real-life application, the proposed system could be compatible with any policy, indicated by the smart contract or left dependent to be statically configured on the Manager's device. On such occasion, the system could adopt a semi-centralized character which may be a more suitable solution depending on the application. However, the proposal of this thesis is to maintain a completely independent and decentralized approach, with blockchain as the governing component. Many more terms and conditions could be added in the smart contract, related to any characteristic of the network's stakeholders such as device specifications e.g., location, vendor, version, system specifications etc., to implement any custom logic. The management would be based on this core and the decisions would be taken around what is true and what is not in comparison with the contract's rules. Figure 7.3 visualizes the flow of the logic from policy to smart contract, along with exemplary content.



1. Only devices from Georgia are allowed to enter the network

2. If any external device requests access to an internal device, it must be registered in USA at least for one year

3. Devices with outdated operating software are not allowed to enter the network

.
.
.

```
pragma solidity ^0.8.0;

contract NetworkAccessControl {
.
.
.
require(isFromGeorgia(msg.sender), "Only devices from Georgia are allowed.");
.
.
.
require(isRegisteredForOneYear(msg.sender), "The device must be registered in the USA for at least one year.");
.
.
.
require(isUpToDateSoftware(msg.sender), "Devices with outdated operating software are not allowed.");
.
.
.
}
```

Figure 7.3: From Policy to Smart Contract Flow

Generally, Ethereum uses a discovery protocol in order for the nodes to find each other. They continuously try to find peers to connect until a specified number of peers is connected. Another way to accomplish the connection, is using Static nodes that are manually connected to each other. This small system follows the latter approach in order to simulate the blockchain based behaviour. It was preferred so unpredictable situations are prevented and better control of the simulation falls under us.

7.3 Components

The main acting components of the system are the constrained IoT devices, the Managers, and the smart contract.

Assumption: This implementation does not involve any simulation for the requesting devices and assumes that their activity is just an arbitrary and asynchronous request to the Manager. IoT devices, that usually belong to Wireless Sensor Networks (WSNs), communicate with the network using the CoAP protocol. The CoAP message is translated then into Remote Procedure Call (RPC) messages to be understandable by the blockchain nodes. Up to this part, every process is excluded from the implementation and is considered as given. The simulation begins since the Manager receives the request and forwards it to the blockchain. There is no networking to be configured between the Manager and the blockchain network, while the whole simulation was developed to run on a local environment.

7.3.1 Local Environment

The basis of our artifact consists of a local blockchain, which is achieved through using Ethereum Virtual Machine (EVM) by running a local blockchain node via the development environment and testing framework of Truffle. EVM was chosen instead of the actual Ethereum blockchain for various practical reasons.

First of all, running a local Ethereum node allows us to control the development and testing environment. We can more easily set up, configure and manage the network according to our specific needs. This benefits us especially in the development phase by enabling faster iterations and debugging. EVM has excellent compatibility with JavaScript, which was used to develop the backend, through specialized libraries.

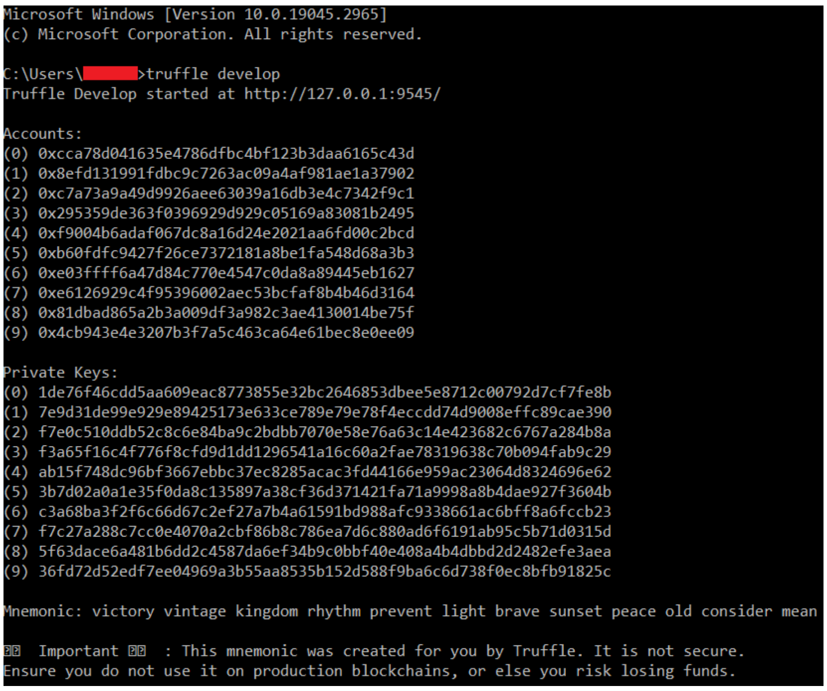
The actual Ethereum network is costly, requiring the use of real Ether (ETH) for transactions and smart contract interactions. This involves costs including gas fees, fact that makes the use of the real platform impractical considering the testing and experimentation purposes we work for. In contrast, a local Ethereum node lets us simulate the behaviour of the Ethereum network without real costs.

Furthermore, EVM is much faster and efficient than interacting with the actual Ethereum network, as it is optimized for local execution. Latency for transactions on local node is much lower and so smart contracts have much faster response times. In this way we win quicker development and testing cycles.

Customizability and control of the network like custom configuration of block times, gas limits and mining difficulty to suit better our specification requirements were also considered when choosing EVM. This let us create scenarios that closely resemble real-world conditions and specific use cases.

Last but not least, local Ethereum node deployment is much simpler and more straightforward process than setting up connections to interact with the Ethereum network. We gained reduced complexity of the development and testing process and thus ease in focusing on the objectives of our research.

When a local node of EVM runs on a device through Truffle environment, Ethereum accounts attached to their corresponding private keys are provided. In our simulation we consider each account to be assigned a participating device. As explained earlier, each device is considered as a blockchain node. In Figure 7.4 we see the Ethereum addresses provided by Truffle environment after initiating the run of a local EVM node.



```
Microsoft Windows [Version 10.0.19045.2965]
(c) Microsoft Corporation. All rights reserved.

C:\Users\[redacted]>truffle develop
Truffle Develop started at http://127.0.0.1:9545/

Accounts:
(0) 0xcca78d041635e4786dfbc4bf123b3daa6165c43d
(1) 0x8efd131991fdbc9c7263ac09a4af981ae1a37902
(2) 0xc7a73a9a49d9926aee63039a16db3e4c7342f9c1
(3) 0x295359de363f0396929d929c05169a83081b2495
(4) 0xf9004b6adaf067dc8a16d24e2021aa6fd00c2bcd
(5) 0xb60fdfc9427f26ce7372181a8be1fa548d68a3b3
(6) 0xe03ffff6a47d84c770e4547c0da8a89445eb1627
(7) 0xe6126929c4f95396002aec53bcfaf8b4b46d3164
(8) 0x81dbad865a2b3a009df3a982c3ae4130014be75f
(9) 0x4cb943e4e3207b3f7a5c463ca64e61bec8e0ee09

Private Keys:
(0) 1de76f46cdd5aa609eac8773855e32bc2646853dbee5e8712c00792d7cf7fe8b
(1) 7e9d31de99e929e89425173e633ce789e79e78f4eccdd74d9008effc89cae390
(2) f7e0c510ddb52c8c6e84ba9c2bdbb7070e58e76a63c14e423682c6767a284b8a
(3) f3a65f16c4f776f8cfd9d1dd1296541a16c60a2fae78319638c70b094fab9c29
(4) ab15f748dc96bf3667ebbc37ec8285acac3fd44166e959ac23064d8324696e62
(5) 3b7d02a0a1e35f0da8c135897a38cf36d371421fa71a9998a8b4dae927f3604b
(6) c3a68ba3f2f6c66d67c2ef27a7b4a61591bd988afc9338661ac6bff8a6fccb23
(7) f7c27a288c7cc0e4070a2cbf86b8c786ea7d6c880ad6f6191ab95c5b71d0315d
(8) 5f63dace6a481b6dd2c4587da6ef34b9c0bbf40e408a4b4dbbd2d2482efe3aea
(9) 36fd72d52edf7ee04969a3b55aa8535b152d588f9ba6c6d738f0ec8bfb91825c

Mnemonic: victory vintage kingdom rhythm prevent light brave sunset peace old consider mean

⚠️  Important ⚠️  : This mnemonic was created for you by Truffle. It is not secure.
Ensure you do not use it on production blockchains, or else you risk losing funds.
```

Figure 7.4: Ethereum Accounts by Truffle

Beyond the blockchain, next comes the backend that is implemented on a local NodeJS express server. The server allows to the developed Application Programming Interface (API) to run locally and hosts its endpoints to receive requests from the frontend interface, an HTML page with scripts of JavaScript, that receives input from the user.

The HTML page is hosted on a local HTTP server and the requests that it invokes reach again the smart contract all the way through the backend, in order to receive and show a response to the user. When the endpoints receive a request, they call the corresponding JavaScript functions that interact with the smart contract on the local blockchain, using the specialized library Web3.

Technical details regarding the implementation of the involved components are more extensively provided in the following subchapters, related to each one separately.

7.3.2 Component: Manager

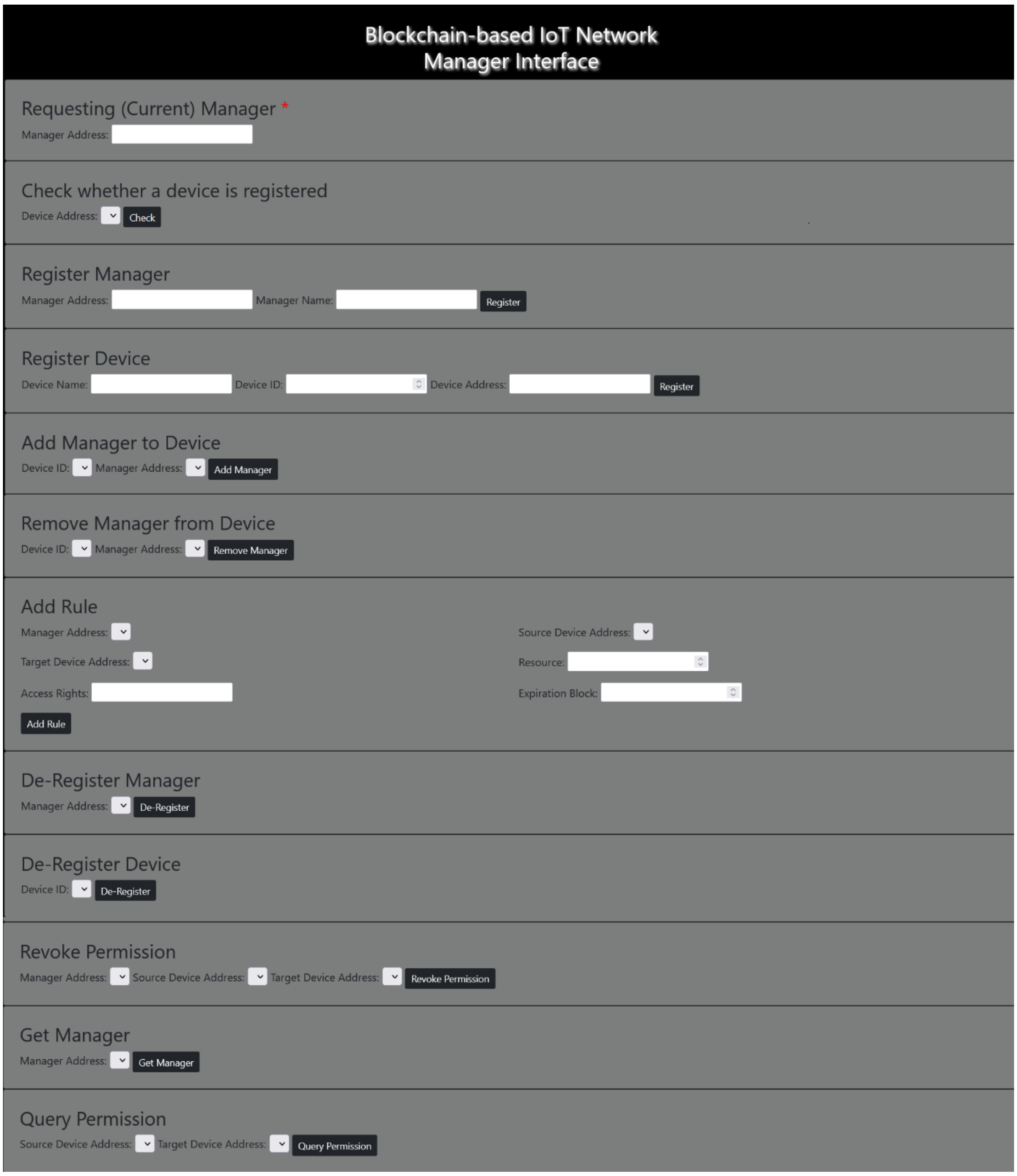
Managers have the main acting role of the three system components. Managers have access to the smart contract's memory and are able to invoke functions that lead to transactions that make changes or make calls in order to retrieve data from the memory, known as queries. The actions performed by the Managers have as follows:

- Register and deregister other Managers to and from the blockchain
- Register and deregister constrained devices to and from the blockchain
- Add and remove rules for the devices under their management to and from the blockchain
- Query the blockchain

The actions are furtherly discussed later in the subchapter of Smart Contract, as the functions that describe its operation.

Managers are responsible for many constrained IoT devices so the latter can actively operate in the network, and so they should be capable of hosting a copy of the blockchain and thus work as full nodes. This features a storage requirement that must be met by the Managers' devices.

Technically, Managers performance is visualised through an HTML form-looking page that invokes JavaScript scripts in order to interact with the backend and with the smart contract by extension. It should be mentioned that the Manager's interface is the point of view of a specific Manager that is specified at the very top of the page as "Requesting (Current) Manager". The Manager, such as any device in the network, has to be assigned an Ethereum address from the pool provided by Truffle. Smart contract-wise, the requesting Manager is the message sender of the transaction or call. This visualization takes place only with purpose to make the concept of the Manager clear and in a more framed real-world application of the system such an interface would not be necessary. The Manager interface is presented in Figure 7.5, where the Manager's available actions are distinguished along.


Blockchain-based IoT Network
Manager Interface

Requesting (Current) Manager *
Manager Address:

Check whether a device is registered
Device Address: Check

Register Manager
Manager Address: Manager Name: Register

Register Device
Device Name: Device ID: Device Address: Register

Add Manager to Device
Device ID: Manager Address: Add Manager

Remove Manager from Device
Device ID: Manager Address: Remove Manager

Add Rule
Manager Address:
Source Device Address:
Target Device Address:
Resource:
Access Rights:
Expiration Block:
Add Rule

De-Register Manager
Manager Address: De-Register

De-Register Device
Device ID: De-Register

Revoke Permission
Manager Address: Source Device Address: Target Device Address: Revoke Permission

Get Manager
Manager Address: Get Manager

Query Permission
Source Device Address: Target Device Address: Query Permission

Figure 7.5: Manager Interface

Note: For usability purposes, for any user that may use this interface, automatically updated dropdown lists are used to provide the available addresses and IDs of each time. Given this, some functions may lose some from their functionality, but the concept remains the same. For example, no check for an unregistered device can take place as in the corresponding dropdown list only the addresses of the registered devices are available.

7.3.3 Component: Smart Contract

The system's operation is governed by the actions that are defined in a specific smart contract. As discussed earlier, every node of the network hosts the smart contract, specifically through their manager, and here is where the term of the distributed application (dApp) appears. The smart contract that was implemented within the current simulation, defines each operation that is allowed by the outlined management system, as it appears in the manager interface. The management function set of the system was influenced by [12] and is shown below in Table 7.1.

Name	Method	Description
Register Manager	Transaction	Registers a Manager into the system
Register Device	Transaction	Registers a Device into the system
Add Manager to Device	Transaction	Registers a Device under a Manager's control
Remove Manager from Device	Transaction	Deregisters a Device from a Manager's control
Add Rule	Transaction	Adds an access control rule under a device, using resource, access rights and access duration (time measured in blocks) as parameters
De-Register Manager	Transaction	De-registers a Manager from the system. Business-wise, the Manager should not manage any device before being removed from the system
De-Register Device	Transaction	De-registers a Device from the system
Revoke Permission	Transaction	Deletes a policy rule from the system
Get Manager	Call	Returns a Manager's information
Query Permission	Call	Checks if a Device can access the resources of another device and returns the access' information
Check whether a device is registered	Call	Checks if a Device is registered and returns its information

Table 7.1: Management Function Set

The smart contract was written in Solidity, the Turing complete programming language supported by the Ethereum platform. As described earlier, the desired management policy was coded into smart contract in Solidity in order to utilize the logic. Each operation of Table

5.1 exists as a function in the developed smart contract and is governed by many restrictions, so the processes follow the proper logic and thus the system's operation makes sense.

As distinguished in Table 7.1, there are two types for a smart contract function. Some functions are triggered by network's transactions incurring fees and delays while others just request queries without cost. Transactions are validated by miners and alter network's state while Calls simply retrieve data from the smart contract's memory. Both types are passages for messages from nodes to the smart contract. An Ethereum contract address is assigned to the smart contract when it is deployed, which is used by the nodes to send the messages to. The processes for both messages, transactions and calls are illustrated in Figure 7.6

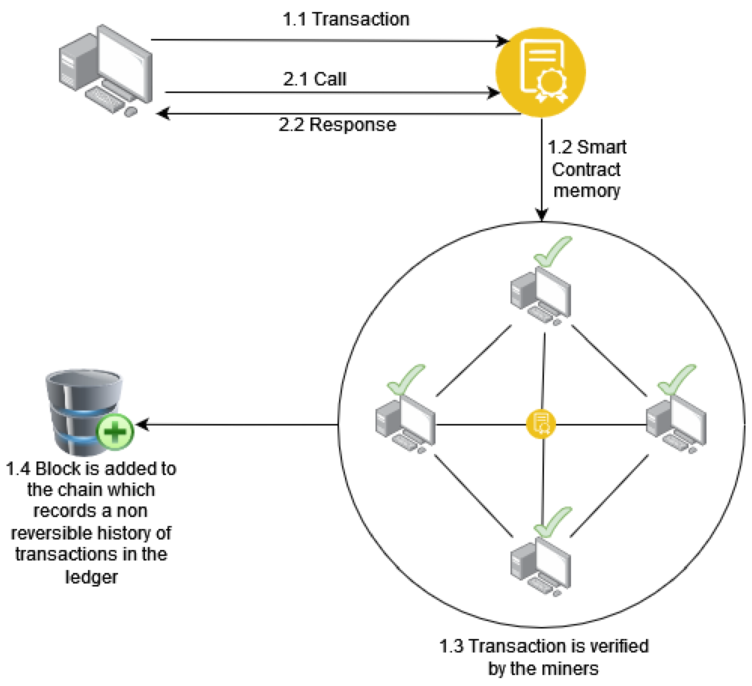

1.1 Transaction
2.1 Call
2.2 Response
1.2 Smart Contract memory
1.4 Block is added to the chain which records a non reversible history of transactions in the ledger
1.3 Transaction is verified by the miners

Figure 7.6: Transaction and Call processes

Chapter 8

Evaluation

This chapter deals with the testing of the system aiming to conclude whether our artifact, the proof-of-concept system that was developed provides an actual solution to the problem as it was defined by the objectives of this thesis. The used methodology DSR, suggests that a comparison between the objectives and the actual results should take place. To do so, the system is tested, and the results are compared to our expectations, so we can evaluate our proposal. There are two types of testing though, Functional and Non-Functional testing and this evaluation takes under consideration both.

8.1 Functional Testing

Functional testing is used to check the system with a business-wise approach, meaning requirements that are more external. Specifically, in functional testing tests the system for bugs and errors and ensures that its functions operate according to plan. As about the bugs and errors, from the smart contract's point of view we should consider the system bug-free since it has made it to run successfully. The Truffle environment that was used for the development of the dApp has built-in compilation system along with an automated testing framework. For the rest system, meaning the frontend and backend components that allow the user to interact with the application, no further testing was conducted to locate errors as it is beyond the scope of this thesis. The web technologies for that aspect were adopted following as many best practices were in knowledge and up to a limit and since the aimed functionality was achieved, no more examination was done. Regarding the proper and expected operation of the system's functions, the methodology of use cases was followed, through various scenarios.

8.1.1 Scenarios

According to different use cases of the system, several management scenarios were invented and run on the developed simulation. The scenarios are completely random but based on rational and possible use cases of such a management system, even in real-life applications. Scenarios are presented below, each along with a brief description and a list of the system functions that need to be activated. Based on the response of the system, we make observations and decide whether the system successfully and correctly satisfied the needs.

Note: For testing with the following scenarios, we will be limited to the 10 given Ethereum addresses, representing 10 network nodes, as they were provided by Truffle environment. Possibility to add more is given, but they are considered enough to prove the functionality of the system.

Assumption: While every function needs a message sender to be called and for that Requesting (Current) Manager is set, when initializing the network and there is no Manager or Devices registered, we assume an initial Boot Manager with standard address that is responsible for letting other Managers and Devices to join the network so it can begin operating. In a real-life application this Boot Manager could be a provided device with hardcoded static address only for serving this purpose.

Scenario 1:

Description:

Devices with names "SONY BRAVIA KD43X72K" and "Apple HomePod 2" and IDs 1 and 2 are registered under Manager "HP Pavilion 14". Device with name "Haier I-Pro Series 5" and ID 3 is registered under Manager "ASUS ROG G16". Device 3 requests through its Manager to check whether Device 1 is registered in the network. At the same time, Device 2 requests through its Manager to get the information of Manager "ASUS ROG G16". The Ethereum address assigned to each device can be seen in Table 8.1.

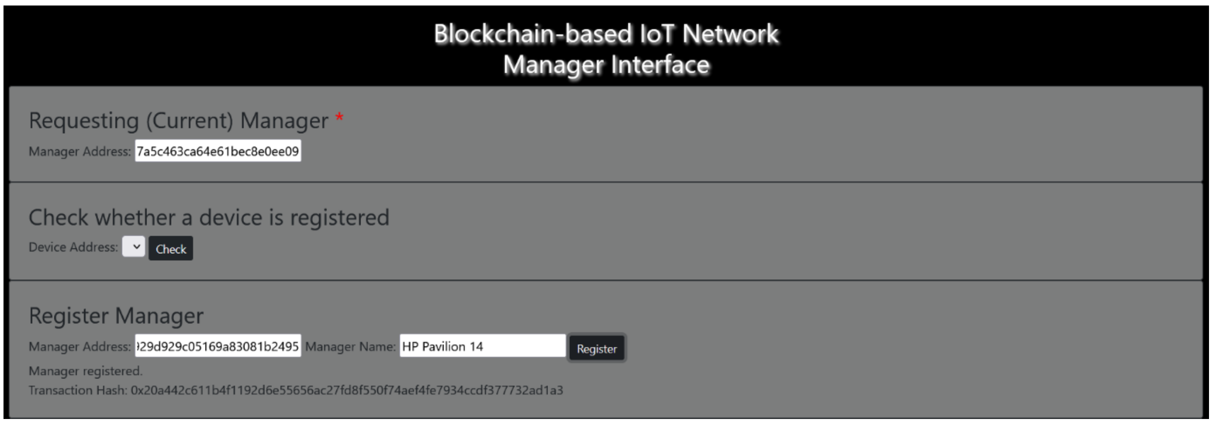
Device Name	Device ID	Device Address
SONY BRAVIA KD43X72K	1	0xcca78d041635e4786dfbc4bf123b3daa6165c43d
Apple HomePod 2	2	0x8efd131991fdbc9c7263ac09a4af981ae1a37902
Haier I-Pro Series 5	3	0xc7a73a9a49d9926aee63039a16db3e4c7342f9c1
HP Pavilion 14	-	0x295359de363f0396929d929c05169a83081b2495
ASUS ROG G16	-	0xf9004b6adaf067dc8a16d24e2021aa6fd00c2bcd
Boot Manager	-	0x4cb943e4e3207b3f7a5c463ca64e61bec8e0ee09

Table 8.1: Ethereum addresses of the involved devices

Actions:

1. Managers have to be registered in the network along with their assigned Ethereum addresses. Function used: Register Manager.
2. Devices have to be registered in the network along with their assigned Ethereum addresses. Function used: Register Device.
3. Device 1 and 2 have to be added Manager “HP Pavilion 14” (0x295359de363f0396929d929c05169a83081b2495). Function used: Add Manager to Device.
4. Device 3 have to be added Manager “ASUS ROG G16” (0xf9004b6adaf067dc8a16d24e2021aa6fd00c2bcd). Function used: Add Manager to Device.
5. Device 3 checks whether Device 1 (0xcca78d041635e4786dfbc4bf123b3daa6165c43d) is registered. Function used: Check whether a device is registered.
6. Device 2 asks for Manager’s “ASUS ROG G16” (0xf9004b6adaf067dc8a16d24e2021aa6fd00c2bcd) information. Function used: Get Manager.

Simulation of the Scenario 1 is illustrated in the following screenshots, presented as Figures:

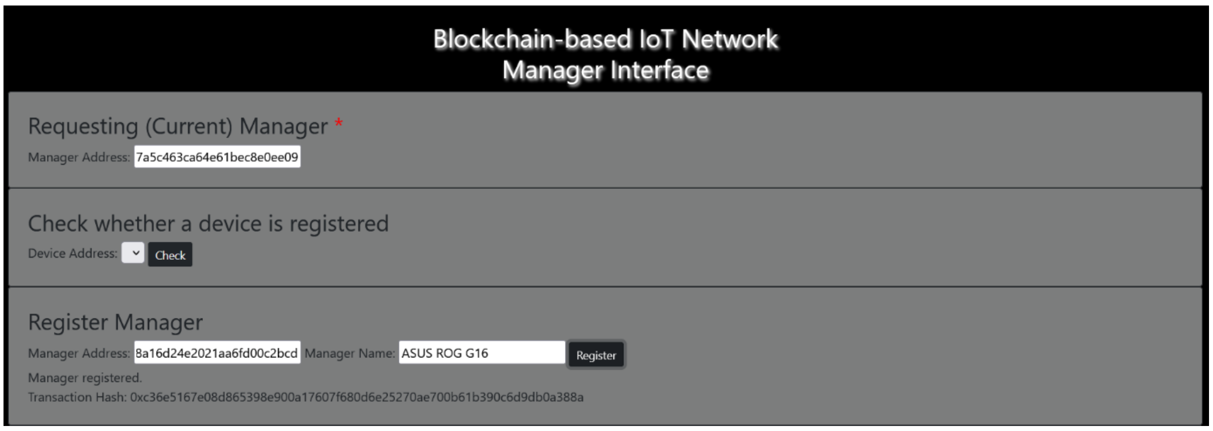

Blockchain-based IoT Network
Manager Interface

Requesting (Current) Manager *
Manager Address: 7a5c463ca64e61bec8e0ee09

Check whether a device is registered
Device Address:

Register Manager
Manager Address: 29d929c05169a83081b2495 Manager Name: HP Pavilion 14
Manager registered.
Transaction Hash: 0x20a442c611b4f1192d6e55656ac27fd8f550f74aef4fe7934ccdf377732ad1a3

Figure 8.1: Register Manager “HP Pavilion 14” using Boot Manager


Blockchain-based IoT Network
Manager Interface

Requesting (Current) Manager *
Manager Address: 7a5c463ca64e61bec8e0ee09

Check whether a device is registered
Device Address:

Register Manager
Manager Address: 8a16d24e2021aa6fd00c2bcd Manager Name: ASUS ROG G16
Manager registered.
Transaction Hash: 0xc36e5167e08d865398e900a17607f680d6e25270ae700b61b390c6d9db0a388a

Figure 8.2: Register Manager “ASUS ROG G16” using Boot Manager

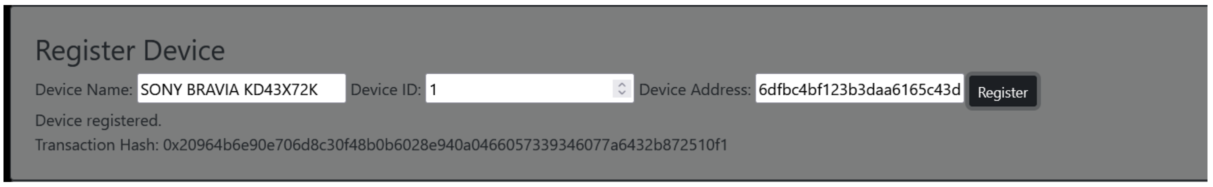

Register Device
Device Name: SONY BRAVIA KD43X72K Device ID: 1 Device Address: 6dfbc4bf123b3daa6165c43d
Device registered.
Transaction Hash: 0x20964b6e90e706d8c30f48b0b6028e940a0466057339346077a6432b872510f1

Figure 8.3: Register “SONY BRAVIA KD43X72K” with ID:1 using Boot Manager

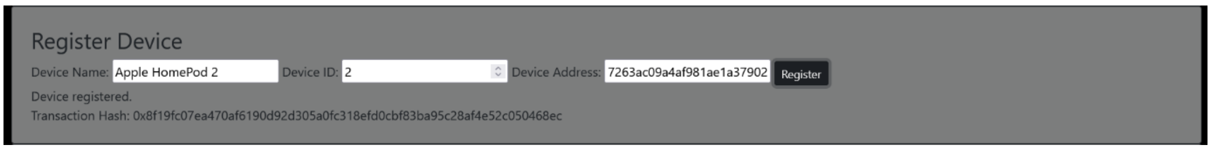

Register Device
Device Name: Apple HomePod 2 Device ID: 2 Device Address: 7263ac09a4af981ae1a37902
Device registered.
Transaction Hash: 0x8f19fc07ea470af6190d92d305a0fc318efd0cbf83ba95c28af4e52c050468ec

Figure 8.4: Register “Apple HomePod 2” with ID:2 using Boot Manager

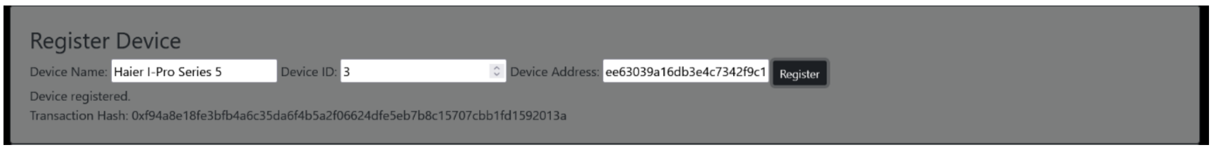
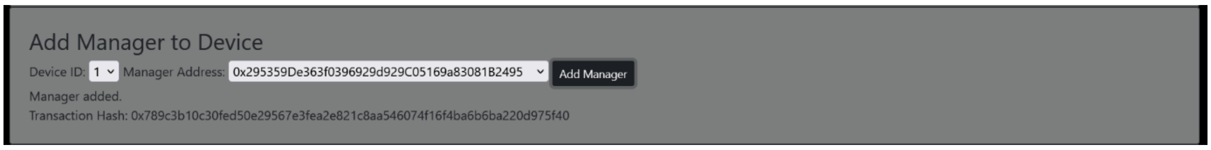

Register Device
Device Name: Haier I-Pro Series 5 Device ID: 3 Device Address: ee63039a16db3e4c7342f9c1
Device registered.
Transaction Hash: 0xf94a8e18fe3bfb4a6c35da6f4b5a2f06624dfe5eb7b8c15707cbb1fd1592013a

Figure 8.5: Register “Haier I-Pro Series 5” with ID:3 using Boot Manager

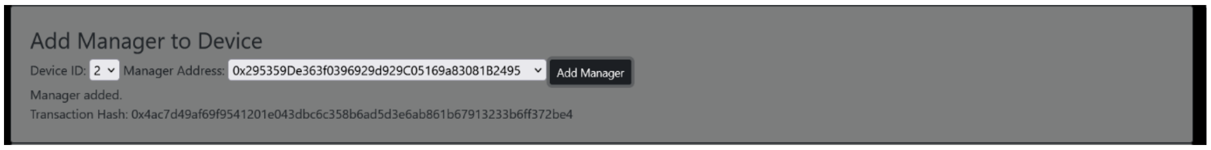

Add Manager to Device

Device ID: 1 Manager Address: 0x295359De363f0396929d929C05169a83081B2495

Manager added.

Transaction Hash: 0x789c3b10c30fed50e29567e3fea2e821c8aa546074f16f4ba6b6ba220d975f40

Figure 8.6: Add Manager “HP Pavilion 14” to Device 1

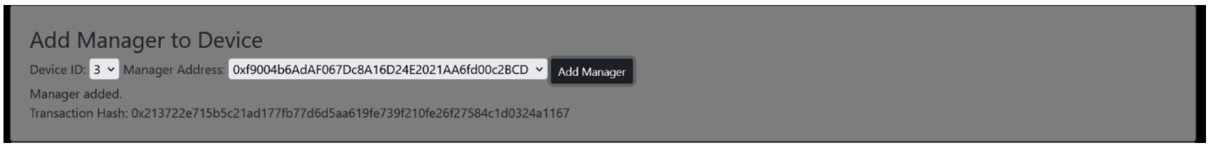

Add Manager to Device

Device ID: 2 Manager Address: 0x295359De363f0396929d929C05169a83081B2495

Manager added.

Transaction Hash: 0x4ac7d49af69f9541201e043dbc6c358b6ad5d3e6ab861b67913233b6ff372be4

Figure 8.7: Add Manager “HP Pavilion 14” to Device 2

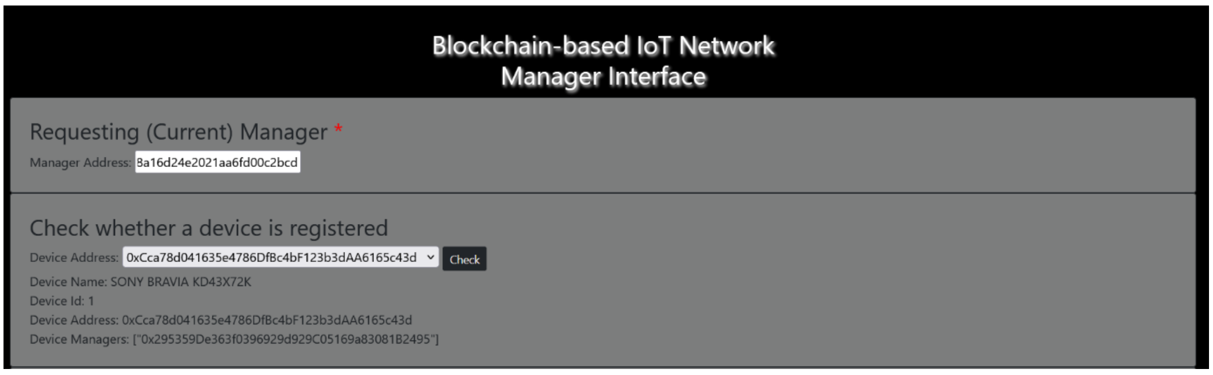

Add Manager to Device

Device ID: 3 Manager Address: 0xf9004b6AdAF067Dc8A16D24E2021AA6fd00c2BCD

Manager added.

Transaction Hash: 0x213722e715b5c21ad177fb77d6d5aa619fe739f210fe26f27584c1d0324a1167

Figure 8.8: Add Manager “ASUS ROG G16” to Device 3


Blockchain-based IoT Network
Manager Interface

Requesting (Current) Manager *

Manager Address: 8a16d24e2021aa6fd00c2bcd

Check whether a device is registered

Device Address: 0xCca78d041635e4786DfBc4bF123b3dAA6165c43d

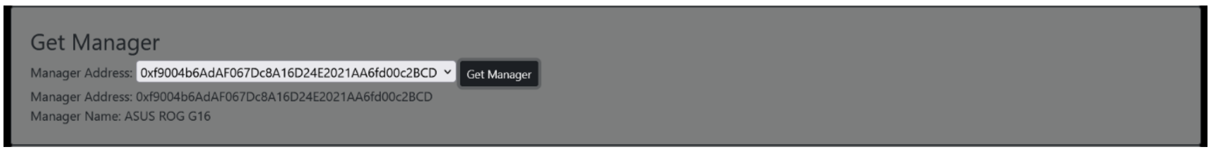
Device Name: SONY BRAVIA KD43X72K

Device Id: 1

Device Address: 0xCca78d041635e4786DfBc4bF123b3dAA6165c43d

Device Managers: ["0x295359De363f0396929d929C05169a83081B2495"]

Figure 8.9: Check whether Device 1 is registered (Device 3) and get its information


Get Manager

Manager Address: 0xf9004b6AdAF067Dc8A16D24E2021AA6fd00c2BCD

Manager Address: 0xf9004b6AdAF067Dc8A16D24E2021AA6fd00c2BCD

Manager Name: ASUS ROG G16

Figure 8.10: Get Manager’s “ASUS ROG G16” information (Device 2)

Observations: Every function was executed successfully. Managers and Devices were normally registered to the network as expected. When Devices requested queries from the system, there information retrieved from the network was correct.

Evaluation: Success

Scenario 2: In continuation to Scenario 1: Device 3 deregisters itself from the network. Another Device with name “SAMSUNG GALAXY S20 FE” and ID 3 is registered and added under the management of Manager “HP Pavilion 14”. This device requests the deregistration of Manager “ASUS ROG G16” and Device 2 requests access to Device 3, on resource 8595, with access rights “read and write” (rw) and expiration block 100. The Ethereum address assigned to new device can be seen in Table 8.2.

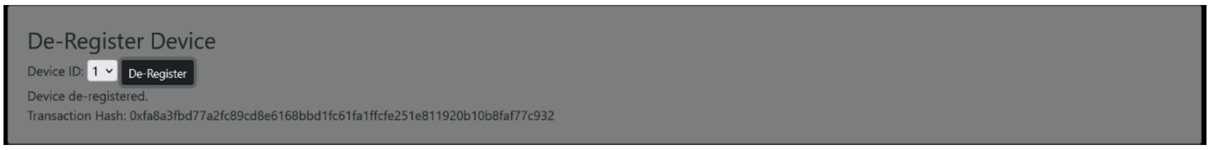
Device Name	Device ID	Device Address
SAMSUNG GALAXY S20 FE	3 (after deregistration of previous device with ID: 3)	0xb60fdfc9427f26ce7372181a8be1fa548d68a3b3

Table 8.2: Ethereum address of the new device

Actions:

1. Deregister Device 3. Function used: De-register Device.
2. The new Device has to be registered in the network with its assigned Ethereum address. Function used: Register Device.
3. Device 3 (new) have to be added Manager “HP Pavilion 14” (0x295359de363f0396929d929c05169a83081b2495). Function used: Add Manager to Device.
4. Device 3 requests deregistration of Manager “ASUS ROG G16” (0xf9004b6adaf067dc8a16d24e2021aa6fd00c2bcd). Function used: De-register Manager.
5. Device 2(0x8efd131991fdbc9c7263ac09a4af981ae1a37902) requests access to Device 3(0xb60fdfc9427f26ce7372181a8be1fa548d68a3b3), on resource 8595, with access rights “read and write” (rw) and expiration block 100. Function used: Add rule.

Simulation of the Scenario 2 is illustrated in the following screenshots, presented as Figures:


De-Register Device

Device ID: 1

Device de-registered.

Transaction Hash: 0xfa8a3fbd77a2fc89cd8e6168bbd1fc61fa1ffcfe251e811920b10b8faf77c932

Figure 8.11: De-register Device 3 (in the screenshot it seems to be Device ID: 1 as selected because the dropdown list was updated after the deregistration of Device 3)

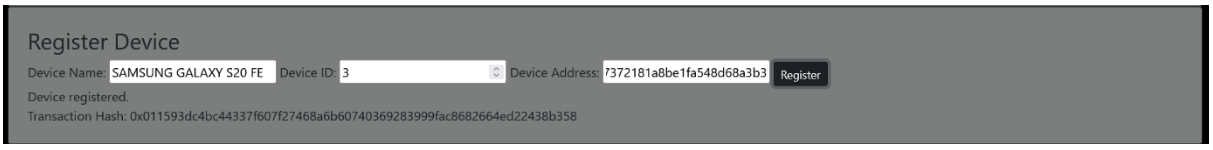

Register Device
Device Name: SAMSUNG GALAXY S20 FE Device ID: 3 Device Address: 7372181a8be1fa548d68a3b3
Device registered.
Transaction Hash: 0x011593dc4bc44337f607f27468a6b60740369283999fac8682664ed22438b358

Figure 8.12: Register “SAMSUNG GALAXY S20 FE” with ID: 3 using Boot Manager

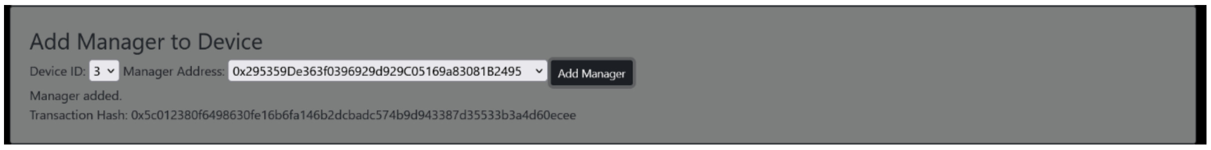

Add Manager to Device
Device ID: 3 Manager Address: 0x295359De363f0396929d929C05169a83081B2495
Manager added.
Transaction Hash: 0x5c012380f6498630fe16b6fa146b2dcbadc574b9d943387d35533b3a4d60ecee

Figure 8.13: Add Manager “HP Pavilion 14” to Device 3

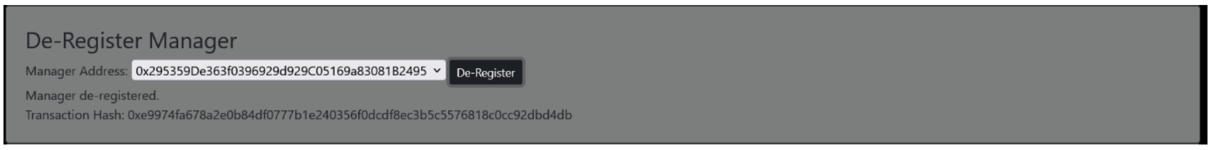

De-Register Manager
Manager Address: 0x295359De363f0396929d929C05169a83081B2495
Manager de-registered.
Transaction Hash: 0xe9974fa678a2e0b84df0777b1e240356f0dcdf8ec3b5c5576818c0cc92dbd4db

Figure 8.14: De-register Manager “ASUS ROG G16” from the network (in the screenshot it seems to be Manager Address: 0x295359de363f0396929d929c05169a83081b2495 as selected because the dropdown list was updated after the deregistration of Manager “ASUS ROG G16”)

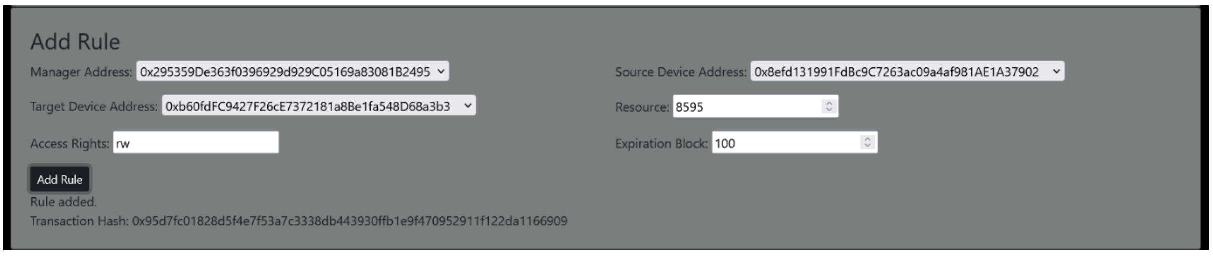

Add Rule
Manager Address: 0x295359De363f0396929d929C05169a83081B2495
Target Device Address: 0xb60fdFC9427F26cE7372181a8Be1fa548D68a3b3
Access Rights: rw
Source Device Address: 0x8efd131991FdBc9C7263ac09a4af981AE1A37902
Resource: 8595
Expiration Block: 100
Rule added.
Transaction Hash: 0x95d7fc01828d5f4e7f53a7c3338db443930ffb1e9f470952911f122da1166909

Figure 8.15: Add rule for access from Device 2 to Device 3

Observations: Every function was executed successfully. Managers and Devices were normally registered and deregistered as expected. When Device 2 asked for access to Device 3, the rule was added.

Evaluation: Successful

Scenario 3: In continuation to Scenario 2: Manager “ASUS ROG G16” re-joins the network. A new Device with name “Sunlight LED” and ID 4 is registered in the network and added under the management of “ASUS ROG G16”. Device 1 asks to see whether there is access from Device 2 to Device 3 and access from Device 2 to Device 4. Then, Device 2 and Device 4 ask for deletion of the permission rule between Device 2 and Device 3. The Ethereum address assigned to new device can be seen in Table 8.3.

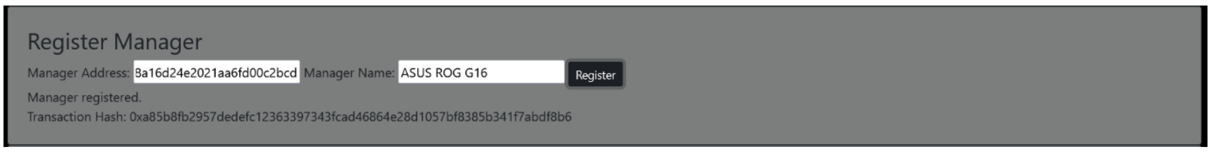
Device Name	Device ID	Device Address
Sunlight LED	4	0xe03ffff6a47d84c770e4547c0da8a89445eb1627

Table 8.3: Ethereum address of the new device

Actions:

1. Manager "ASUS ROG G16" has to be re-registered in the network with its assigned Ethereum address. Function used: Register Manager.
2. The new Device has to be registered in the network with its assigned Ethereum address. Function used: Register Device.
3. Device 4 has to be added Manager "ASUS ROG G16"(0xf9004b6adaf067dc8a16d24e2021aa6fd00c2bcd). Function used: Add Manager to Device.
4. Device 1 queries permission between Device 2(0x8efd131991fdbc9c7263ac09a4af981ae1a37902) and Device 3 (0xb60fdfc9427f26ce7372181a8be1fa548d68a3b3). Function used: Query Permission.
5. Device 1 queries permission between Device 2(0x8efd131991fdbc9c7263ac09a4af981ae1a37902) and Device 4(0xe03ffff6a47d84c770e4547c0da8a89445eb1627). Function used: Query Permission.
6. Device 2 and Device 4 ask for permission revoke between Device 2(0x8efd131991fdbc9c7263ac09a4af981ae1a37902) and Device 3(0xb60fdfc9427f26ce7372181a8be1fa548d68a3b3) (simultaneously). Function used: Revoke permission.

Simulation of the Scenario 3 is illustrated in the following screenshots, presented as Figures:



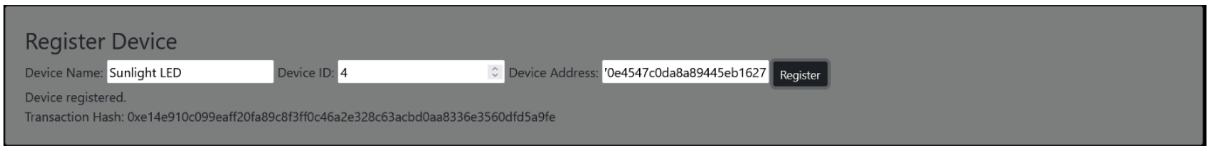
Register Manager

Manager Address: 8a16d24e2021aa6fd00c2bcd Manager Name: ASUS ROG G16

Manager registered.

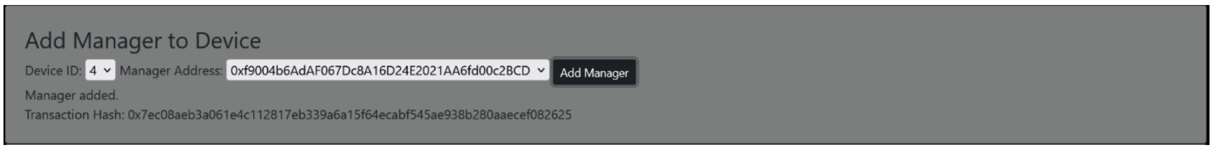
Transaction Hash: 0xa85b8fb2957dedefc12363397343fcad46864e28d1057bf8385b341f7abdf8b6

Figure 8.16: Register Manager "ASUS ROG G16" using Boot Manager


Register Device
Device Name: Sunlight LED Device ID: 4 Device Address: '0e4547c0da8a89445eb1627

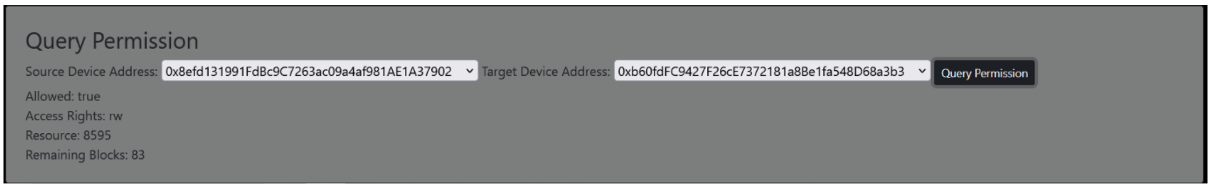
Device registered.
Transaction Hash: 0xe14e910c099eaff20fa89c8f3ff0c46a2e328c63acbd0aa8336e3560dfd5a9fe

Figure 8.17: Register Device 4 using Boot Manager


Add Manager to Device
Device ID: 4 Manager Address: 0xf9004b6AdAF067Dc8A16D24E2021AA6fd00c2BCD

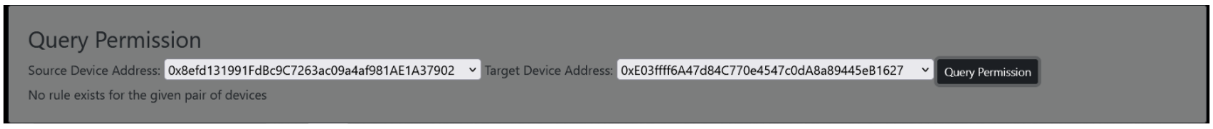
Manager added.
Transaction Hash: 0x7ec08aeb3a061e4c112817eb339a6a15f64ecabf545ae938b280aaecef082625

Figure 8.18: Add Manager “ASUS ROG G16” to Device 4


Query Permission
Source Device Address: 0x8efd131991FdBc9C7263ac09a4af981AE1A37902 Target Device Address: 0xb60fdFC9427F26cE7372181a8Be1fa548D68a3b3

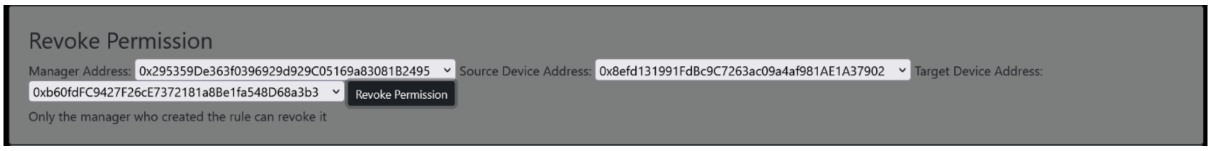
Allowed: true
Access Rights: rw
Resource: 8595
Remaining Blocks: 83

Figure 8.19: Query permission between Device 2 and Device 3 (Device 1)


Query Permission
Source Device Address: 0x8efd131991FdBc9C7263ac09a4af981AE1A37902 Target Device Address: 0xE03ffff6A47d84C770e4547c0dA8a89445eB1627

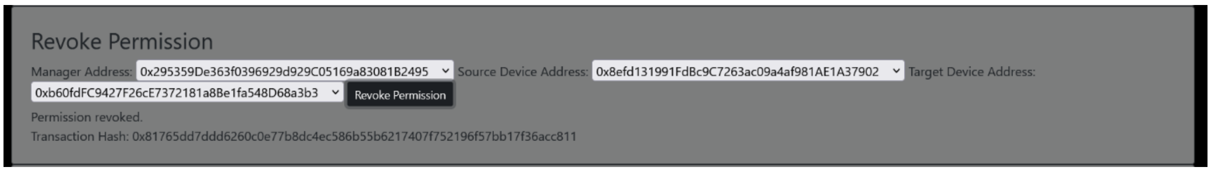
No rule exists for the given pair of devices

Figure 8.20: Query permission between Device 2 and Device 4 (Device 1)


Revoke Permission
Manager Address: 0x295359De363f0396929d929C05169a83081B2495 Source Device Address: 0x8efd131991FdBc9C7263ac09a4af981AE1A37902 Target Device Address: 0xb60fdFC9427F26cE7372181a8Be1fa548D68a3b3

Only the manager who created the rule can revoke it

Figure 8.21: Revoke permission between Device 2 and Device 3 (Device 4)


Revoke Permission
Manager Address: 0x295359De363f0396929d929C05169a83081B2495 Source Device Address: 0x8efd131991FdBc9C7263ac09a4af981AE1A37902 Target Device Address: 0xb60fdFC9427F26cE7372181a8Be1fa548D68a3b3

Permission revoked.
Transaction Hash: 0x81765dd7ddd6260c0e77b8dc4ec586b55b6217407f752196f57bb17f36acc811

Figure 8.22: Revoke permission between Device 2 and Device 3 (Device 2)

Observations: Every function was executed successfully. Managers and Devices where registered normally as expected. When Device 1 performed queries on the permission between devices, system responded correctly that access permission is allowed from Device 2 to Device 3 along with its specifications and that no rule exists from Device 2 to Device 4. Also, when Device 4 asked to revoke the permission from Device 2 to Device 3 the request was declined because Device’s 4 Manager is not the one that created the rule (as indicated by the policy of the network). Instead, when Device 2 asked to revoke the permission, it did successfully as its Manager is the one that created the rule.

Evaluation: Successful

The above scenarios cover the whole spectrum of system’s functionality. We observed that each function worked as expected and we can describe the outcome as successful. Therefore, functional testing of our application was successful too.

8.2 Non-functional Testing

Non-functional testing tests functional aspects of the system such as Scalability, Performance, Mobility, Concurrency and Security. This kind of testing can be thought more important than functional testing here, since these characteristics are more substantial considering this thesis’ aim, to provide a solution that is applicable on a real-life global scale environment. In this subchapter, we discuss about the non-functional testing that was carried out for the system, covering some of the mentioned properties.

8.2.1 Hardware

The machine that was used to perform the non-functional testing of the system is a NodeJS server running on *64-bit Microsoft Windows 10 Home* operating system. The processor is an *Intel i7-6700HQ* with 4 multithread cores and maximum frequency 3.50 GHz. The RAM is 16.0 GB.

8.2.2 Testbed Settings

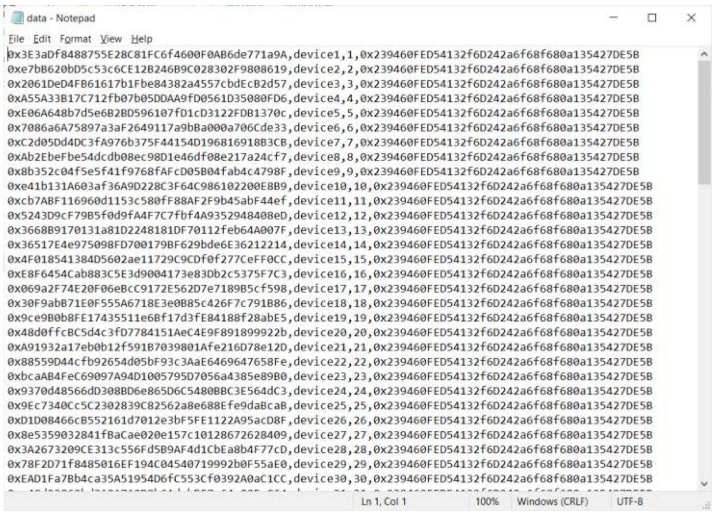
Our small-scale prototype is implemented on a private blockchain network on EVM, thus it is evaluated there as well. As said earlier, there are two types of node discovery in Ethereum. A dynamic one where a protocol is used to detect the peers and a static one where the connections are manually configured. The fact that we adopted the second method is the reason why we needed to assume the Boot Managers in subchapter 7.1 to let the new nodes join and discover their peers. In functional testing, in the scenarios that were checked, we needed to configure 1 Boot Manager, we had another 2 Managers involved and 5 IoT devices in total. But these settings have no impact to our results neither to functional nor to non-functional testing, as each of these entities is virtual and not configured differently in the network. For example, if we set some Managers to act as miners too, then the network's parameters would be affected. However, as this this is a simulation and we used private Ethereum network where the consensus mechanism is typically not implemented, we were satisfied as we had a small group of nodes that trust each other. This is an admission within this thesis, that the proof of concept does not consider security, but this requires only a configuration to do so. For this reason, there is going to be a noticeable difference on the non-functional results that we obtain, with the more realistic implementation. Our prototype may not capture the full complexity and scale of the actual Ethereum network, but it still allows us to simulate the interactions and test various scenarios within the limitations of our local setup. So, our simulation can provide a relative representation of a decentralized network and the results that we obtain mainly in the non-functional testing have a relevance to the properties of the live environment of public Ethereum network, although its complexity is reduced.

8.2.3 Performance

The performance of the system plays a huge role in the value of its implementation in a real environment. Response time is a usual metric to clearly describe the performance of a system. As in the case of our simulation, where response times of one of its functions were used in order to determine how well it performs. To do so, we recorded the response time of the API calls in increasing number of simultaneous requests to one of the system's functions.

More specifically, a tool called Apache JMeter was used to create a test plan to the system's API backends. The tool allows to configure API calls specifying any variable parameter each time, even with .csv files, and to create threads for independent simultaneous calls on the application called "users". After the settings are done a run of the test plan can be executed and the results can be recorded even as a graph, as the tool provides built-in automatic graph creation.

Firstly, scripts were written to generate a .csv file with the API parameters for 100 new devices to be registered in the network. Each line represented a device in the following form: deviceAddress,deviceName,deviceID,currentManagerAddress. We can see how the file looked like in Figure 8.23.



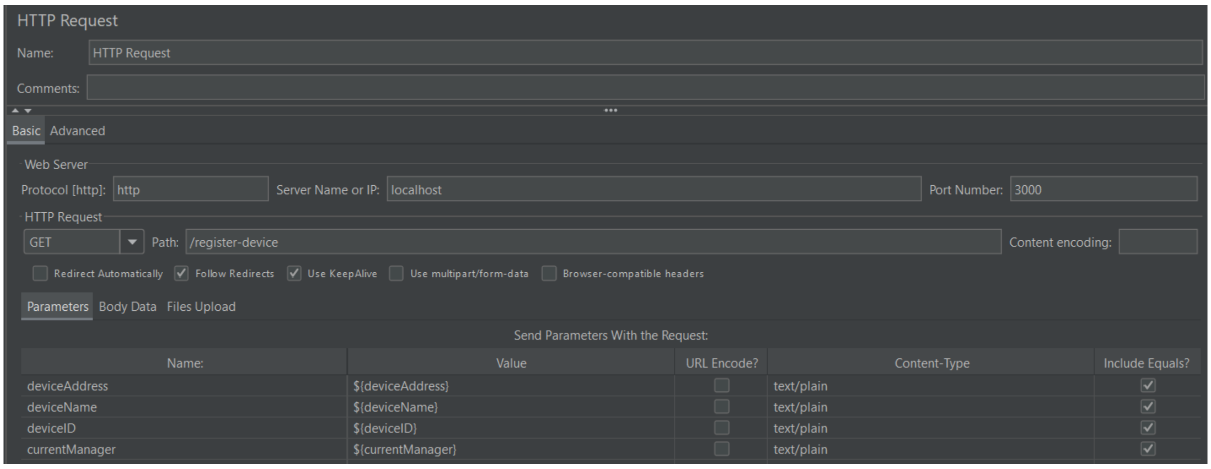
data - Notepad

```
0x3E3aDf8488755E28C81FC6f4600F0AB6de771a9A,device1,1,0x239460FED54132f6D242a6f68f680a135427DE5B
0xe7bB620bD5c53c6CE12B246B9C028302F9808619,device2,2,0x239460FED54132f6D242a6f68f680a135427DE5B
0x2061DeD4FB61617b1Fbe84382a4557cbdEcB2d57,device3,3,0x239460FED54132f6D242a6f68f680a135427DE5B
0xA55A33B17C712fb07b05DDAA9fD0561D35080FD6,device4,4,0x239460FED54132f6D242a6f68f680a135427DE5B
0xE06A648b7d5e6B2BD596107fD1cD3122FDB1370c,device5,5,0x239460FED54132f6D242a6f68f680a135427DE5B
0x7086a6A75897a3aF2649117a9bBa000a706Cde33,device6,6,0x239460FED54132f6D242a6f68f680a135427DE5B
0xC2d05Dd4DC3fA976b375F44154D196816918B3CB,device7,7,0x239460FED54132f6D242a6f68f680a135427DE5B
0xAb2EbeFbe54dcdb08ec98D1e46df08e217a24cf7,device8,8,0x239460FED54132f6D242a6f68f680a135427DE5B
0x8b352c04f5e5f41f9768fAFcD05B04fab4c4798F,device9,9,0x239460FED54132f6D242a6f68f680a135427DE5B
0xe41b131A603af36A9D228C3F64C986102200E8B9,device10,10,0x239460FED54132f6D242a6f68f680a135427DE5B
0xcb7ABF116960d1153c580fF88AF2F9b45abF44ef,device11,11,0x239460FED54132f6D242a6f68f680a135427DE5B
0x5243D9cF79B5f0d9fA4F7C7fbf4A9352948408eD,device12,12,0x239460FED54132f6D242a6f68f680a135427DE5B
0x3668B9170131a81D2248181DF70112feb64A007F,device13,13,0x239460FED54132f6D242a6f68f680a135427DE5B
0x36517E4e975098FD700179BF629bde6E36212214,device14,14,0x239460FED54132f6D242a6f68f680a135427DE5B
0x4F018541384D5602ae11729C9CDf0f277CeFF0CC,device15,15,0x239460FED54132f6D242a6f68f680a135427DE5B
0xE8F6454Cab883C5E3d9004173e83Db2c5375F7C3,device16,16,0x239460FED54132f6D242a6f68f680a135427DE5B
0x069a2F74E20F06eBcC9172E562D7e7189B5cf598,device17,17,0x239460FED54132f6D242a6f68f680a135427DE5B
0x30F9abB71E0F555A6718E3e0B85c426F7c791B86,device18,18,0x239460FED54132f6D242a6f68f680a135427DE5B
0x9ce9B0b8FE17435511e6Bf17d3fE84188f28abE5,device19,19,0x239460FED54132f6D242a6f68f680a135427DE5B
0x48d0ffcBC5d4c3fD7784151AeC4E9F891899922b,device20,20,0x239460FED54132f6D242a6f68f680a135427DE5B
0xA91932a17eb0b12f591B7039801Afe216D78e12D,device21,21,0x239460FED54132f6D242a6f68f680a135427DE5B
0x88559D44cfb92654d05bF93c3AaE6469647658Fe,device22,22,0x239460FED54132f6D242a6f68f680a135427DE5B
0xbcaAB4FeC69097A94D1005795D7056a4385e89B0,device23,23,0x239460FED54132f6D242a6f68f680a135427DE5B
0x9370d48566dD308BD6e865D6C5480BBC3E564dC3,device24,24,0x239460FED54132f6D242a6f68f680a135427DE5B
0x9Ec7340Cc5C2302839C82562a8e688Efe9daBcaB,device25,25,0x239460FED54132f6D242a6f68f680a135427DE5B
0xD1D08466cB552161d7012e3bF5FE1122A95acD8F,device26,26,0x239460FED54132f6D242a6f68f680a135427DE5B
0x8e5359032841fBaCae020e157c10128672628409,device27,27,0x239460FED54132f6D242a6f68f680a135427DE5B
0x3A2673209CE313c556Fd5B9AF4d1CbEa8b4F77cD,device28,28,0x239460FED54132f6D242a6f68f680a135427DE5B
0x78F2D71f8485016EF194C04540719992b0F55aE0,device29,29,0x239460FED54132f6D242a6f68f680a135427DE5B
0xEAD1Fa7Bb4ca35A51954D6fC553Cf0392A0aC1CC,device30,30,0x239460FED54132f6D242a6f68f680a135427DE5B
```

Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8

Figure 8.23: Generated .csv file

Then the test plan was run for HTTP requests on the .csv file on the URL of our application, hosted locally, for the corresponding function call to register new devices. For 100 devices, 100 calls API calls took place. The Apache JMeter setting looked as in Figure 8.24.



HTTP Request

Name: HTTP Request

Comments:

Basic | Advanced

Web Server

Protocol [http]: http   Server Name or IP: localhost   Port Number: 3000

HTTP Request

GET   Path: /register-device   Content encoding:

☐ Redirect Automatically   ☑ Follow Redirects   ☑ Use KeepAlive   ☐ Use multipart/form-data   ☐ Browser-compatible headers

Parameters | Body Data | Files Upload

Send Parameters With the Request:

Name:	Value	URL Encode?	Content-Type	Include Equals?
deviceAddress	${deviceAddress}	☐	text/plain	✓
deviceName	${deviceName}	☐	text/plain	✓
deviceID	${deviceID}	☐	text/plain	✓
currentManager	${currentManager}	☐	text/plain	✓

Figure 8.24: Apache JMeter configuration

An initial run for the registration of the devices was conducted first. The obtained results in the form of graph are presented in Figure 8.25.

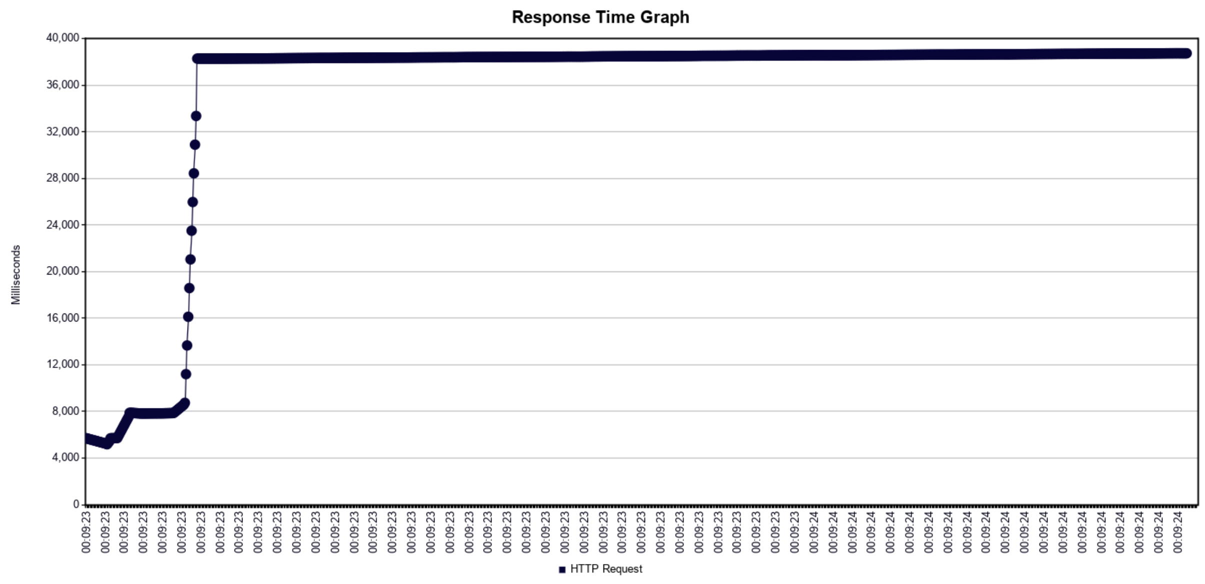


Figure 8.25: Response time (ms) of the API call to register devices as function of increasing number of calls

Then, the test plan was executed another time on the same devices, meaning that this time the system would reject by reverting their double registration, as policy of the network. The corresponding results are presented in the form of graph in Figure 8.26.

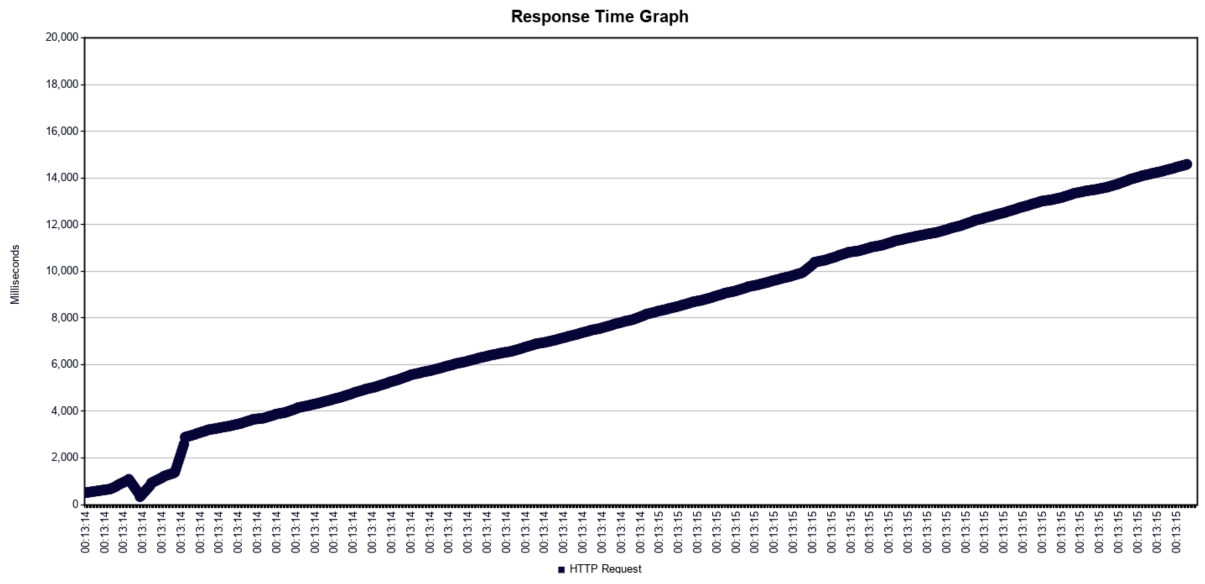


Figure 8.26: Response time (ms) of the API call to register devices as function of increasing number of calls – Revert

From the two graphs, we can make the following observations:

1. In any of the two cases, we can see that the system performs satisfactorily, considering the expectations or standards of the industry from such a system. The

response time in both graphs is fluctuating approximately to half a minute or less, for such a high number of device registrations. Comparing the system's response times with conventional systems would provide more valuable context and insights for its performance.

2. In the first graph, where devices were registered, the response times are greater than in the second graph when there was revert. This indicates a relation between the kind of the process that is caused from the call and the response time. The process of invoking a smart contract’s function is more demanding in time than revert, as expected.
3. In the first graph the response times are rapidly increased and then stabilized on a specific value with just a negligible increase as calls become more. In the second graph, the response time is linearly increasing. This indicates that the process of a smart contract’s invoked function may be dependent on other parameters, that rise its response time to a certain value, and then and then increases very little due to the factor of increased calls. In contrast, the process of revert seems to be not dependent on other factors and only the increased demand in calls rise its response time.

Summarizing this small experiment, we can extract some value from this proof of concept, as it seems to respond successfully to an increasing load of requests. The scenario under which the requests were conducted was more realistic as JMeter allowed simultaneous threads to touch them off. Additionally, we should mention the probable dependence of smart contract’s functions’ response time from other factors than increasing load, which rise it to higher values and in a way stabilize it there. Also, the obvious difference in response time between a function that was invoked and a function that was reverted, should not be unnoticed, with the latter to has much lesser than the other. We could be more precise in our conclusions about performance if we compared our results to corresponding ones of a conventional IoT management system. We could also be if we tested more functions of the system. In such a case we would be able to compare the response time between functions with different complexity and make conclusions about the relevance between response time and complexity.

8.2.4 Scalability

Scalability can also be considered as one of the high-priority requirements of the system. It is the ability of the system to handle increased workloads or growing demands while maintaining or improving its performance, efficiency, and reliability. Since we aim our proof-of-concept to take global dimensions, we have to take scalability under serious consideration.

Based on the results of the experiment that took place within subchapter 7.2.5 to check performance, we can draw conclusions about the scalability of the prototype too. This is because we tested the system in increasing workload, such as a growing number of devices or simultaneous requests. Under these conditions we observed that the system maintained acceptable performance. We saw that the response times remained relatively stable or increased linearly with the workload. This suggests that the system is handling the increased workload in a scalable manner.

8.3 Security

The implemented system, as well as blockchain-based decentralized applications in general, faces various vulnerabilities that threaten their safety and security. This subchapter discusses some of these issues.

8.3.1 Denial of Service with failed call

External calls in smart contracts may fail accidentally because of programming errors or intentionally by intruders. If many calls in a single transaction are executed using a loop can prevent nodes from interacting with the smart contract [6]. EVM has a stack for calls limited up to 1024. Regarding the proof of concept that uses EVM, in such a case of repeating calls, it is even possible to reach the stack's depth limit and completely control the smart contract.

8.3.2 Denial of Service with gas limit

In Ethereum, each block has an upper bound of gas that can be spent for computation. If the gas spent exceeds that limit, the system is lead to DoS and the transaction fails. When there are arrays of unknown size that increases over time and there is a loop over them, we should plan it to take multiple blocks and thus multiple transactions. If every computation takes place in a single transaction, there is a high risk of DoS and this becomes extremely dangerous when an intruder can manipulate the amount of needed gas [6].

8.3.3 Re-entrancy

Re-entrancy attack is also known as recursive call attack and refers to the case where a malicious contract calls back into the original one before the invocation of the function is done.

Different invocations of the same function may interact with each other maliciously. In this way, an attacker can execute withdrawals in repetition while letting his balance stay unaffected [7].

8.3.4 Sybil Attack

Distributed applications do not adopt any user identity verification measure. Users can just create accounts, having the possibility to create as many as they will. If a user gets the chance to act as many simultaneously, means that attackers can insert into the network many clients under their control, with result to be able to manipulate which blocks are refused or not in their favour [7].

8.3.5 Malicious Managers

Within the proof of concept, the term of Managers was described as hubs between the constrained IoT devices with limited capabilities, and the blockchain network. It was explained that Managers do not take decisions but only forward the messages to the smart contract. But still, a malicious Manager could drop transactions or even send wrong ones. Here comes a trust assumption, that Managers are trustworthy, presumably configured by the owners of the IoT network. Of course, there are many measures to control this vulnerability, such as mechanisms to ensure the integrity of transactions by the Managers like cryptographic techniques or digital signatures and access control mechanisms to restrict Managers' privileges. These securities though, are kept beyond the scope of this thesis.

8.4 Limitations

We have seen from the previous chapters and subchapters that in many aspects the system is limited. Some limitations may arise from the implementation itself while some others from the nature of the technologies that were used. Since the setup varies from application to application, and as the subject of assessment of the thesis is the system as a concept in terms of technologies and not of execution, we will focus on the limitations that govern the technology behind, in context of public application of the system. In more detail, they are described in the next paragraphs.

8.4.1 Block time

If our implementation is escalated into the public Ethereum network, it will have to face its standard limitations, along with the time that it is needed to create a block, meaning the time it waits in the queue to be picked up and the time the miner needs to verify it. Ethereum block time is currently around 12 seconds. If any IoT device needs to urgently access another node through the management system, this would be a great issue.

8.4.2 System changes

System allows transactions based on a specific smart contract's policy. If this policy changes or the logic in which the system deals with these transactions, the system will have to be rebuilt all the way from the beginning. Gas costs are likely to incur, as well as the smart contract's memory will be reset, and its data lost.

8.4.3 Time measurement

Our proposal uses block number to give the sense of time, so access is expired after a defined duration. But in a real-life application this wouldn't be a case. IoT management systems use the measurement units that people use too in their everyday lives. System would have to become more complex to use the sense of time within the blockchain application.

Chapter 9

Discussion

By conducting this thesis, we set from the very beginning the objective to provide a secure and scalable up to a global level access management solution for constrained IoT devices, that is also independent from any central authority. We specifically defined some questions to be answered.

1. Can we develop a blockchain-based IoT device management system to operate at a worldwide scale?

Within this thesis we utilised blockchain technology for device identity authentication, device access management, transaction processing and data protection and storage in a decentralized manner. The distributed nature of the blockchain technology facilitates for achieving this decentralization. There are many applications that can be implemented out of this model, to satisfy a wide range of needs in the industry market.

2. How does blockchain technology benefit IoT?

1. Equal stakes: The great impact of blockchain in the economic sector is more than obvious. We have all seen the wide spread of cryptocurrencies into the society, in our everyday lives. Implementations such as Bitcoin and Ether, as they were investigated in the background information provided by this thesis, have been adopted by a vast group of businesses and individuals as commercial currency. In the same way, blockchain can be implemented to IoT networks to allow to billions of devices to communicate without any intermediary infrastructure. At the same time, utilizing blockchain will take under consideration the issue of owner and authority for each stakeholder, making sure that the shares are equally given, resulting thus to equal stakes.

2. Security: Blockchain technology offers a strong and reliable framework for safeguarding IoT devices and data. Its decentralized structure guarantees that information stored on the ledger cannot be tampered with or altered without proper authorization. This fortifies the integrity and credibility of IoT systems, minimizing the chances of data breaches and unauthorized entry.

3. Decentralization: The concept of decentralization is a key feature enabled by blockchain technology in the context of IoT. It eradicates the necessity for a central authority or middleman, allowing IoT devices to communicate and interact directly with one another. This eradication of centralized servers and intermediaries has significant benefits, including improved system resilience, scalability, and the elimination of vulnerable single points of failure.

4. Data Integrity and Transparency: The transparency and decentralized characteristics of blockchain contribute to improved data integrity and transparency within IoT networks. Each transaction or data modification is meticulously documented on the blockchain, generating a traceable and verifiable history of events. This fosters accountability and guarantees that all involved parties have access to precise and current information.

5. Trust and Identity Management: Blockchain technology offers a robust and decentralized solution for managing trust and identities within the realm of IoT. By leveraging the blockchain, each IoT device can possess a distinctive digital identity, which is securely recorded on the decentralized ledger. This approach ensures the authenticity of devices and facilitates the implementation of efficient access control mechanisms. Through blockchain-based identity management, trust is fostered among devices, enabling secure and reliable interactions within IoT networks.

Chapter 10

Conclusion and Future Work

The ultimate goal of this thesis was to research and analyse in the maximum possible depth the blockchain technology and investigate whether and how its employment within the access management of IoT networks of constrained devices was feasible. We managed to design and create a small-scale proof of concept of such a system, intended for operation on the Ethereum platform, in a global range. To achieve this, extensive research was required in order to choose the most proper and suitable technology. Within the limited, small-scale system that we aimed to, the offered choices were also limited to satisfy its requirements correspondingly, and given this, an additional consideration was taken. Although, the smooth flow of research development in the developed system wasn't affected at all. Instead, the outcome of the system's evaluation was positive, proving that a blockchain-based IoT management system may excel beyond the conventional management systems in IoT networks. We have repeated the benefits from the use of blockchain technology many times along the paper's extent.

When evaluating the system, we also came across some limitations, which however have to do with the blockchain technology or even the technologies that the system used in general and not with the implementation itself. For example, if in a real-life application, in an urgent case where a device requests access in hurry and the system takes 12 seconds, Ethereum's average block creation time, or even more to respond, a big issue will be caused against the client. Limitations like this, regarding the non-functional properties of the system such as Performance and Scalability, were discussed in detail. These issues have to be solved so we are let to apply the system at a greater scale, on the real and public Ethereum network. Concluding, although blockchain technology is still relatively new in terms of development, there are so many benefits to be gained with its application, offered for a great spectrum of the industry sectors. When the technical barriers are successfully faced, blockchain is promising for changing to the better a vast number of systems in the near future.

Regarding any future elaboration of the current work, it would be of great interest to evaluate the present proof of concept in advance. More performance tests could be executed, to be more precise and correct about the related observations and conclusions which have a lot to say about the value of the accomplished work. For example, more functions could be tested through API calls in order to obtain information about the comparison between the performance and the complexity of the functions.

To furtherly work on this research, it would be worth the effort to see the performance of the system under real conditions too. More specifically, to test the properties such this one, when the invented technology is applied on actual constrained IoT devices that operate as physical nodes. Our prototype treated the nodes as virtual components with an Ethereum account assigned to them, on a private EVM node that runs locally through the environment of Truffle. Consequently, the whole setup may have proven the functionality and the utilization of the theory and been an honest and relevant representation but as mentioned in Chapter 7, Evaluation, it can’t reflect the real data. If real data were obtained, the real-world implications of the system could be clarified.

Citations

[1] Bosamia, Mansi & Patel, Dharmendra. (2018). Current Trends and Future Implementation Possibilities of the Merkel Tree. INTERNATIONAL JOURNAL OF COMPUTER SCIENCES AND ENGINEERING. 6. 294-301. 10.26438/ijcse/v6i8.294301.

[2] Boudko, Svetlana & Leister, Wolfgang. (2019). Building Blocks of Negotiating Agents for Healthcare Data. 10.1145/3366030.3366108.

[3] Brocke, Jan vom & Hevner, Alan & Maedche, Alexander. (2020). Introduction to Design Science Research. 10.1007/978-3-030-46781-4_1.

[4] Buterin, Vitalik. (2015). On Public and Private Blockchains. Ethereum Foundation Blog, accessed 22 April 2023. <https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains>

[5] Dexter, Shawn. (2018). Cryptographic Puzzle – Understanding Proof of Work. Mango Research, accessed 26 April 2023. <https://www.mangoresearch.co/cryptographic-puzzle-proof-of-work>

[6] Kushwaha, Satpal & Joshi, Sandeep & Singh, Dilbag & Kaur, Manjit & Lee, Heung-No. (2022). Systematic Review of Security Vulnerabilities in Ethereum Blockchain Smart Contract. IEEE Access. PP. 1-1. 10.1109/ACCESS.2021.3140091.

[7] Liu, Zheng & Zhu, Hongyang. (2022). Fighting Sybils in Airdrops. 10.48550/arXiv.2209.04603.

[8] Maxie, Emily. (2018). Pros and Cons of the Delegated Proof-of-Stake Consensus Model. Very Technology Blog, accessed 24 April 2023. <https://www.verytechnology.com/iot-insights/pros-and-cons-of-the-delegated-proof-of-stake-consensus-model>

[9] Nakamoto, Satoshi. (2008), Bitcoin: A Peer-to-Peer Electronic Cash System. Bitcoin Org., accessed 23 April 2023. <https://bitcoin.org/bitcoin.pdf>

[10] Natoli, Christopher & Yu, Jiangshan & Gramoli, Vincent & Veríssimo, Paulo. (2019). Deconstructing Blockchains: A Comprehensive Survey on Consensus, Membership and Structure.

[11] Nelaturu, Keerthi & Du, Han & Đức Phong, Lê. (2022). A Review of Blockchain in Fintech: Taxonomy, Challenges, and Future Directions. Cryptography. 6. 18. 10.3390/cryptography6020018.

[12] Novo, Oscar. (2019). Scalable Access Management in IoT Using Blockchain: A Performance Evaluation. 6. 4694-4701. 10.1109/JIOT.2018.2879679.

[13] Ouaddah, A. & Mousannif H. & Kalam, A. A. E., & Ouahman, A. A. (2017). Access control in the Internet of Things: Big challenges and new opportunities. Computer Networks, 112, 237-262.

[14] Ouaddah, A. & Kalam, A. A. E., & Ouahman, A. A. (2016). FairAccess: A new blockchain-based access control framework for the Internet of Things. Security and Communication Networks, 9(18), 5943-5964.

[15] Paul, Anup & Qu, Xin & Wen, Zheng. (2021). Blockchain–a promising solution to internet of things: A comprehensive analysis, opportunities, challenges and future research issues. Peer-to-Peer Networking and Applications. 14. 1-26. 10.1007/s12083-021-01151-0.

[16] Pinno, O. J. A. & Gregio, A. R. A., & Bona, L. C. E. D. (2017). ControlChain: Blockchain as a central enabler for access control authorizations in the IoT. In Proceedings of the IEEE Global Communications Conference (GLOBECOM), 1-6.

[17] Rao, Suhas & Chendanda, Devaiah & Deshpande, Chetan & Lakkundi, Vishwas. (2015). Implementing LWM2M in constrained IoT devices. 10.1109/ICWISE.2015.7380353.

[18] Sayeed, Sarwar & Marco-Gisbert, Hector & Caira, Tom. (2020). Smart Contract: Attacks and Protections. IEEE Access. PP. 1-1. 10.1109/ACCESS.2020.2970495.

[19] Smith, Corwin. (2023). Transactions. Ethereum Org., accessed 25 April 2023. <https://ethereum.org/en/developers/docs/transactions/>

[20] Szabo, Nick. (1994). Smart Contracts. University of Amsterdam, accessed 25 April 2023.<https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html>

Appendix A

This appendix provides information about the tools and technologies used within this thesis.

Truffle Framework

Development framework for Ethereum that provides several tools to help with developing, compiling and deploying smart contracts. It was used to run a local blockchain network on Ethereum EVM, so then the smart contract created for the proof of concept was deployed and we could interact with it in a development environment.

Solidity

Programming language made specifically for writing smart contracts on Ethereum. It is commonly used to define how smart contracts behave and the policy that governs them. We used Solidity to serve this purpose, as the smart contract is responsible for controlling all the rules and operations of our blockchain-based IoT management system.

HTML

Hypertext Markup Language, is a standard language for creating structure and content for webpages. Within the thesis it satisfied the need for the design and structure of the Manager's interface. HTML allowed us to define the layout, elements and components of the user interface of our application.

JavaScript

A widely-used programming language that brings interactivity and dynamic features to web applications. It was used to implement the dynamic features of Manager's application. This involved procedures such as API calls, handling user interactions and updating the front-end dynamically based on data from the back-end.

Web3

A JavaScript library that provides an interface for interacting with smart contracts based on Ethereum. It lets the connection to an Ethereum node, sending transactions and interacting with smart contracts using JavaScript code. Within the thesis, we utilized Web3 to establish a connection between the back-end application and the smart contract hosted on the local blockchain node. In this way we achieved communication and interaction with the contract.

Node.js

Open source, server-side JavaScript runtime environment. It allows the execution of JavaScript code on the server rather in a web browser. It also provides an event-driven architecture and a non-blocking I/O model, making it well-suited for building scalable and efficient web application. It was used as the runtime environment for our back-end application, that handled the processing and execution of the server-side logic. This involved defining routes, handling API requests, interacting with the smart contract and the local blockchain node and performing necessary computations and data manipulations.

Apache JMeter

Open-source tool used for testing the performance and scalability of web applications. It helped us to conduct non-functional testing of the system, with focusing specifically on performance. It allowed the simulation of multiple requests happening simultaneously and therefore the measurement of the system’s response time under different loads. In this way we could obtain valuable insights into the performance characteristics of the system.

Appendix B

This appendix provides a manual for running the system, including each step that has to be followed, detailed.

System Requirements:

- Operating System: Windows, Linux, or macOS
- Node.js and npm installed (version 12 or higher)
- Truffle Framework installed (version 5 or higher)

Step 1: Download the System Files

1. Download the system files provided to you as a ZIP (available from https://github.com/ageorg47/dissertation.git)
2. Extract the files to a directory of your choice.

Step 2: Install Dependencies

3. Open a terminal or command prompt.
4. Navigate to the root directory of the system files (\dissertation-main).
5. Run the following command to install the necessary dependencies:

```
npm install
```

Step 3: Run the Local Blockchain Node

6. Open a terminal or command prompt.
7. Navigate to the backend directory of the system files (\backend).
8. Run the following command to start the Truffle development network:

```
truffle develop
```

9. Once the development network is running, you will see a list of Ethereum addresses. Take note of these addresses as they will be needed later.

Step 4: Compile and Deploy the Smart Contract

10. In the same terminal or command prompt, run the following command to compile and deploy the smart contract:

```
migrate
```

Step 5: Start the Backend Server

11. Open a new terminal or command prompt.
12. Navigate to the backend directory of the system files (\backend).
13. Run the following command to start the backend server:

```
npm run dev
```

Step 6: Start the Frontend Server

14. Open another terminal or command prompt.
15. Navigate to the frontend directory within the system files (\frontend).
16. Run the following command to start the frontend server:

```
http-server
```

Step 7: Access the Manager's Interface

17. Open a web browser.
18. Enter the following URL in the address bar:

```
http://127.0.0.1:8080
```

19. The Manager's interface page should now be displayed.