Leonor Maria Cabrita Augusto

Licenciada em Ciências da
Engenharia Eletrotécnica e de Computadores

An application of blockchain smart contracts and IoT in logistics

Dissertação para obtenção do Grau de Mestre em

Engenharia Eletrotécnica e de Computadores

Orientador: Ricardo Jardim-Gonçalves, Professor Catedrático, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Co-orientador: Ruben Costa, Investigador, UNINOVA Centro de Tecnologia e Investigação

Júri

Presidente:	Dr. José Manuel Matos Ribeiro da Fonseca - FCT/UNL
Arguente:	Dr. Carlos Eduardo Dias Coutinho - ISCTE/IUL
Vogal:	Dr. Ruben Duarte Dias da Costa - UNINOVA

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Junho, 2019

An application of blockchain smart contracts and IoT in logistics

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Aos meus avós,

Acknowledgements

Gostaria de agradecer ao meu orientador, o Professor Ricardo Gonçalves, por me ter aceite para este projeto e guiado ao longo da elaboração da dissertação. Ao meu co-orientador, Doutor Ruben Costa, por ser sempre optimista, por me disponibilizar o seu tempo e conhecimentos e por me estabelecer prazos, sem as quais eu provavelmente não estaria a entregar este documento a tempo.

Aos meus colegas de curso e amigos, com quem não só partilhei mágoas e sucessos académicos, mas também algumas festas memoráveis que não trocaria por nada.

À minha família. Aos meus pais, Zé Augusto e Nicolina, por me terem apoiado, inspirado e ensinado as coisas mais importantes que sei. À Edite, que ainda hoje olha por mim. À minha irmã Joana, por vezes mais mãe do que irmã, mas sempre a minha melhor amiga, e amável como mais ninguém. Ao meu cunhado Filipe, que já passou a irmão, e aos meus sobrinhos, Artur, Joaquim e Amélia, que adoro. Ao João, com quem partilho as melhores memórias que tenho, e com quem quero fazer muitas mais.

Abstract

Over the last few years, interest has emerged in blockchain, a decentralized ledger technology (DLT) created for use in cryptocurrencies, but with a great potential to be used in other application domains. One of them is supply chain management, tracking and tracing, which are key processes to the logistics industry, made difficult due to the lack of standards or trust between actors, miscommunication, fraud and bureaucratic delays, among other issues. In order to overcome some of these challenges, the solution presented in this dissertation proposes a blockchain system application created with Ethereum smart contracts technology. Its main purpose is to be used in supply chain and logistics for the tracking and tracing products, where the storage of important data is done and verified in a trustworthy, decentralized system. The technical solution presented here implements methods for tracking, certification, quality control and authentication, and integrates the communication of blockchain with IoT devices, which play an important role in monitoring products and automating these processes.

This approach is validated by the development of a smart contract system and two browser-based applications to interact with it. The first application allows users to access and view their product's tracking data, while the second bridges the communication between an Arduino UNO microcontroller collecting temperature readings and our smart contract system. The work presented here highlights the benefits of these technologies applied to logistics and validates the feasibility of this approach, ultimately giving insight into the capabilities, qualities, but also of the limitations a system like this can have.

Keywords: blockchain, logistics, smart contracts, supply chains, IoT, Ethereum

Resumo

Ao longo dos últimos anos, um crescente interesse tem surgido em *blockchain*, uma tecnologia de registro descentralizado (DLT) criada para aplicação em sistemas de criptomoedas, mas com um grande potencial para ser usada em outros tipos de aplicações. Uma delas é na gestão e seguimento de cadeias de fornecimento, que são processos-chave no setor de logística, frequentemente dificultados pela falta de normas ou confiança entre diferentes entidades, pela má comunicação, fraudes e atrasos burocráticos, entre outros problemas. Para superar alguns destes desafios, a solução apresentada nesta dissertação propõe uma aplicação de um sistema *blockchain*, criado com a tecnologia de *smart contracts* (contratos inteligentes) de Ethereum. O principal objetivo desta solução é ser utilizada para o seguimento e rastreamento de produtos ao longo de cadeias de fornecimento em logística, sendo o armazenamento de dados importantes feito e verificado num sistema confiável e descentralizado. A solução técnica apresentada aqui implementa métodos de seguimento, certificação, controlo de qualidade e autenticação, integrando também a comunicação de blockchain com dispositivos IoT, que desempenham um papel importante na monitorização de produtos e na automatização destes processos.

Esta abordagem é validada pelo desenvolvimento de um sistema de *smart contracts* e duas aplicações web para interagir com ele. A primeira aplicação permite aos utilizadores do sistema acederem e visualizarem os dados de seguimento dos seus produtos, enquanto a segunda faz a ponte de comunicação entre um microcontrolador Arduino UNO que recolhe leituras de temperatura e as envia para nosso sistema de *smart contracts*. O trabalho apresentado aqui pretende destacar os benefícios destas tecnologias aplicadas à logística, validando a viabilidade da abordagem tomada e, em última análise, dando uma visão das capacidades, qualidades, mas também das limitações que um sistema como este pode ter.

Palavras-chave: blockchain, logística, smart contracts, cadeias de fornecimento, IoT, Ethereum

Contents

List of Figures

Glossary

blocks	Data structures containing the information being stored in the blockchain. When they are added to the blockchain, they are immutable and irreversible .
consensus	The system by which users participating in the blockchain network come to agreement on a block's validity and if it can be added to the existing chain .
DLT	A type of data structure that resides across multiple devices. Both blockchain and smart contracts are examples of DLTs. They are generally comprised of three main components: a data model that captures the present state of the ledger, a language of transactions to change the ledger state, and a system of consensus or protocol by which the participants of the ledger agree on transactions .
double spending	A potential flaw in a digital cash scheme where the same single digital token is spent more than once .
finality	If a blockchain system's consensus mechanism does not permit forking, the blockchain is said to have finality .
forking	Happens when two valid blocks are added to the chain almost simultaneously creating a fork (divergence) in the blockchain .
nodes	Users of a blockchain network are commonly referred to as its nodes .
peer-to-peer	P2P networks are decentralized systems where the users of the network both provide its foundation and participate in it. Each user, or peer, provides computing resources to the network, enabling other peers to use it for performing whatever tasks the network is meant to do - per example, data transfers or transactions. The resources given by a peer can include data storage, processing power or network bandwidth .

permissioned Refers to a type of blockchain network. To join a permissioned blockchain, one needs be authorized/checked before credentials for using the network are given. All members of the network therefore are known, which makes the entry of malicious actors much more unlikely. The counterpart to permissioned blockchains are public or permissionless ones (ex. bitcoin) .

Acronyms

BLE	Bluetooth Low Energy.
DLT	Decentralized Ledger Technology.
EOC	End-Of-Chain.
EVM	Ethereum Virtual Machine.
GSM	Global System for Mobile communications.
IoT	Internet-of-Things.
P2P	Peer-to-peer.
PoEt	Proof-of-Elapsed-time.
PoW	Proof-of-Work.
RBAC	Role-Based Access Control.
RPC	Remote Procedure Call.
SC	Smart Contract.

Chapter 1

Introduction

1.1 Motivation

1.1.1 Decentralizing trade

Over the last few years, there has been an increased interest in the development of decentralized systems and computing architectures. With the advent of the internet, improved communications and processing power have fed the tendency to decentralize computing tasks - from storage, to processing, and nowadays also networks[1].

However, while decentralized networks have existed for many years in the form of peer-to-peer (P2P) systems, due to security issues they could never be applied to processes that involve high-stake transactions. It was the invention of the blockchain, a Decentralised Ledger Technology (DLT) used in securing P2P networks, that has started to change that paradigm [2].

The origins of the blockchain, as well as its first application in the bitcoin currency, are by now a well known subject to many technological and financial institutions. In 2008 Satoshi Nakamoto launched the bitcoin cryptocurrency and with it the first application of a DLT named blockchain emerged[3].

The great revolution bitcoin brought was creating a viable, secure and transparent system to log and execute transactions, without the need for a trusted third-party intermediary, like a bank, to prevent the problem of double spending or other malicious actions. This solution could only be achieved with the invention of blockchain, a technology that uses a cryptographic protocol system named proof-of-work(PoW) for reaching consensus within a P2P network. It is with this protocol that the security of transactions is ensured. As the decentralised network increases in size and number of users, so does its security.

By having most members of the network contributing to its maintenance, the need for a third party that oversees both the network and it's transactions is eliminated - hence the

denomination of blockchains as decentralized systems [4] [5]. This decentralization of power in transactions greatly improves their security, because by having the intermediary eliminated there is no longer a single point of attack for putting any transactions in jeopardy [4].

Security and decentralization are important points of value for blockchain technology, but they are not the only ones. Transparency and immutability are other important characteristics of benefit in certain blockchain applications [6].

In fact, recently blockchain technology has gained attention as it has been increasingly noted that this immutable ledger system can be used outside of the cryptocurrency spectrum. It can be used to agilize bureocracy, create trust through transparency and give easy access to trustworthy information. The Gartner Hype Cycle for Emerging Technologies of 2017 identified blockchain as one of technologies that will lead to reformation in whole industries, having its breakthrough development during the next five years [7].

An important piece that powered the interest rise on blockchain and its potential was the invention of smart contracts. Smart contracts were first successfully implemented in blockchain by Vitalin Buterik in 2014, via the Ethereum platform [8], being essentially protocols that verify, secure and enact transactions or agreements between consenting parties in a decentralized network. They enable developers to create applications that adhere to the most varied sets of rules for ownership, transactions and state transitions, permitting also the decentralized storage and management of data.

In Ethereum technology specifically, smart contract code is stored in the blockchain. To interact with it, users execute function calls in the form of transactions, which can then change the smart contract's state. Smart contracts are therefore comparable to state machines that exist within a blockchain. For a more in-depth explanation see Section 2.2.1 of the state of the art.

Smart contracts technology's versatility allows for more complex applications to be built on top of blockchain, which is why over the years many different ones have indeed been studied and to some extent implemented. Applications for public notaries, authentication of individuals, objects or documents, anti-counterfeit mechanisms and decentralized storage are some of the most popular uses of blockchain that have been explored by companies for varied purposes [9]. Worthy of note is that these are all very common and necessary mechanisms for managing large chains of value, such as the ones found in the logistics sector [4].

1.1.2 Challenges in logistics

The logistics sector, one of the most important in today's global economy, consists of the many entities that make their business creating and maintaining supply chains of goods. In most European countries, this sector's revenue represents between 6 to 10% of each country's GDP, (see Figure 1.1.) Worldwide, the logistics sector's projected growth is to hit 15.5tn[1] USD dollars by 2023 (almost doubling it's value in a period of eight years) [10]. However, in spite of its tremendous value in today's global economies, it is remarkable how this industry is fraught with risk and plagued by sources of avoidable cost [11].

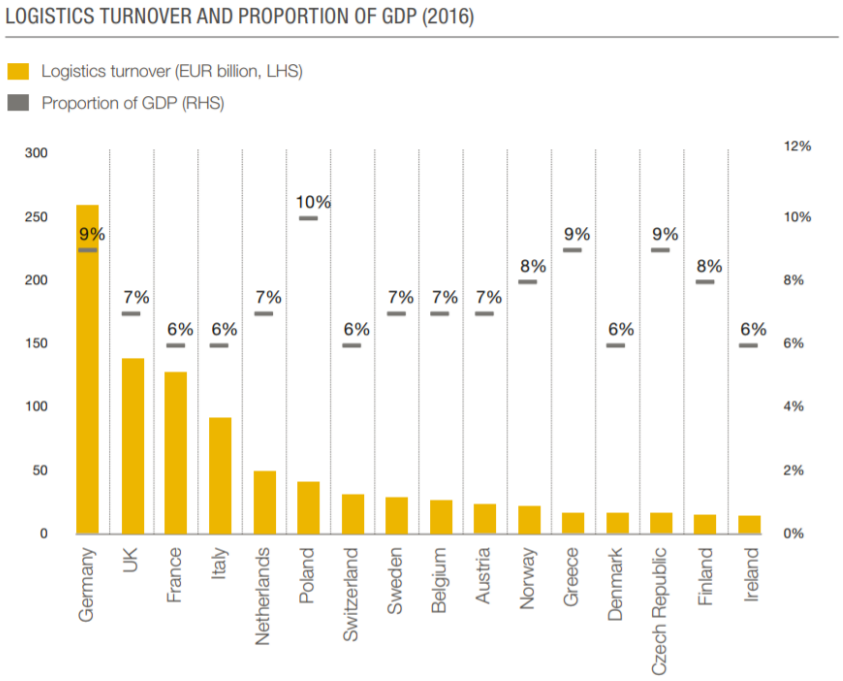


Figure 1.1: Logistics revenue (ranging between 25 - 260bn €) in relation to EU countries percentage of GDP (between 6-10%) in 2016. From [12].

A supply chain encompasses many processes, from conception to delivery of products. Transport, storage and manufacture are guaranteed by several separate entities, between whom there is often a lack of established trust [11]. The intervenients in this network are tasked with creating a product and then delivering it to consumers per their demand - a task of high complexity, and that due to many different factors is rarely completed optimally. Figure 1.2 illustrates the most basic intervenients in the supply chain process. Real supply chains are rarely so simple, involving not just one but many of these parties working in tandem to supply one product.

Executing supply chains is not an easy task, and in fact there are many difficulties companies go through when navigating this field. The industry is notably fragmented; it involves many different companies working together, and yet they rarely share data

[1] The short scale large number naming system is used in all instances of this dissertation on which it can be applied, including this one referring to "trillions".

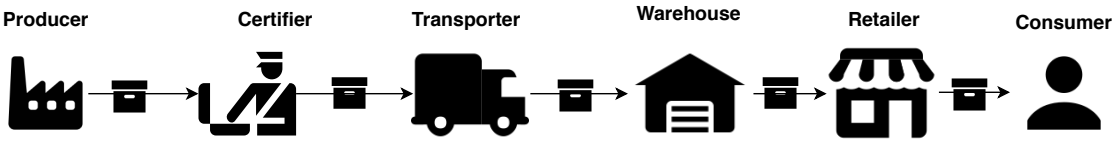


Figure 1.2: Diagram of a basic supply chain tracking process of a product, and its main intervenients.

and keep their own records and databases private. It doesn't help that there is a lack of common standards that facilitate their cooperation. Additionally, regulatory authorities have mandatory systems in place that still rely on paper-based documentation, and issues like theft, fraud, smuggling, counterfeit and the like are commonplace and difficult to tackle [13].

So with the aforementioned issues in mind, it can be theorized that the logistics industry would benefit from a number of improvements, namely transparency amongst its participants, standardization of processes, modernization of regulatory practices and an overall greater level of security and authentication. All of these requirements are of the kind that blockchain technology can address. Because logistics is an industry based on transactions where high flow of information occurs, and because many of its problems stem from issues in transaction-making that blockchain directly addresses, it is our belief that the implementation of decentralised networks to aiding logistics operations could have significant impact in their improved security, speed and transparency, while enabling greater trust to exist between trading parties [14].

We are not the only ones to have noticed how well blockchain could fit with these processes. Currently, there are already a number of large companies exploring how to integrate blockchain into logistics operations - namely Walmart, UPS, FedEx, DHL and Maersk [15] [13] [6]. The interest of industry giants on this new technology can be taken as another indicator of its potential in field.

Many start-ups exploring the use of blockchain for tracking, transparency and validation in supply chains have also emerged - like Provenance [16] or Faizod [17], which aim to integrate blockchain for tracking products through QR code labels or RFID, respectively. In fact the use of Internet-of-Things (IoT) solutions in these systems is a common proposal.

IoT in logistics refers to monitoring and facilitating supply chain processes by establishing communication between supply chain management systems and sensors or actuating devices. These are attached or in the vicinity of the items that are being processed, and can send and receive data through the established communication channel. Their data collection and actuation capabilities can be integrated into these management systems, particularly for the purposes of tracking items, ensuring their integrity and detecting issues in the supply chain.

Integrating IoT technologies in supply chain tracking systems therefore adds further

value to any innovative proposals, including ones that contemplate blockchain in their solutions.

With this outlook in mind, we believe there are many improvements a blockchain-based system could bring to making transactions in logistics, specifically:

1. **Transparency for consumers** - Accessing a product's history and origin from an immutable, secure database informs and strengthens customers' trust on their purchase choices [16].

2. **Traceability of products** - While enabling transparency and giving customers, regulators and auditors access to important information, traceability guarantees that products whose history cannot be traced in the network are easily identified as potential cases of fraud or contraband. This is not only crucial to identifying and solving issues quickly, but also a valuable tool to gain a customer's trust when products go on sale.

3. **Security in transactions** - All the intervening parties in the supply chain benefit from the added security a decentralised blockchain system can bring to registering transactions and certifications whilst also enforcing authentication.

4. **Diminished bureaucratic delays** - It is possible to facilitate and improve bureaucracy by using secure smart contracts to complete signing and verification processes that might otherwise waste a lot of time and money [6]. Having a platform for this that isn't reliant on paper also helps issues like documentation being damaged or lost.

5. **Identifying faults in the supply chain** - Collecting data on supply chain processes can lead to quicker identification and solving of issues, resulting in a more efficient and profitable business which ultimately benefits customers and clients as well.

1.2 Proposed Solution

In light of the issues facing logistics and the opportunities brought about with new distributed ledger technologies, this dissertation forms a proposal of a blockchain-based application for use in logistics, with particular focus on the tracking and tracing of a product's journey throughout a supply chain. The system envisioned is designed and implemented using smart contract technologies, and is aimed at serving the different kinds of entities that participate in the logistics industry, whilst also being a platform that can provide costumers at the end-of-chain information on the items they are buying.

1.2.1 Research Question and Hypothesis

This dissertation's work tries to answer a few key research questions in the field of blockchain logistics applications, namely:

- Can blockchain and smart contracts technologies be used as a basis for supply chain management and tracking of goods?
  - Can they handle the complexities required by a system like this, particularly in terms of storage and computation requirements?
  - Can they agilize transactional and certification processes?
  - Are these applications ready for real-world use?
- What approaches can be taken to seamlessly integrate IoT devices and blockchain technologies in the process of tracking items through supply chains?

The ensuing hypothesis is that a system built with blockchain and smart contract technologies can successfully process supply chain tracking data and achieve levels of trustworthiness, transparency and security that are superior to the ones found in centralized systems. By having all entities participating in the supply chain process use the same blockchain and smart contracts application for storing tracking data and handling important tasks such as authentication and quality checks, the communication and management of bureaucracy associated with logistics can be improved. At the same time, some of the steps in these processes are facilitated by introducing IoT device data into the blockchain system and have it evaluated by smart contracts. These technologies can be made available via user-friendly APIs, and can potentially overcome their most common issues to become usable in real systems.

1.2.2 A smart contract system for supply chain tracking

A simplified view of the steps involved in product tracking when using our blockchain system are laid out in Figure 1.3. It shows how a product can be tracked from its origin by having its data continuously updated and its status automatically evaluated and approved via smart contracts and IoT device data input. The data collected can help managers identify issues in supply chains, and provide proof of origin and traceability for items.

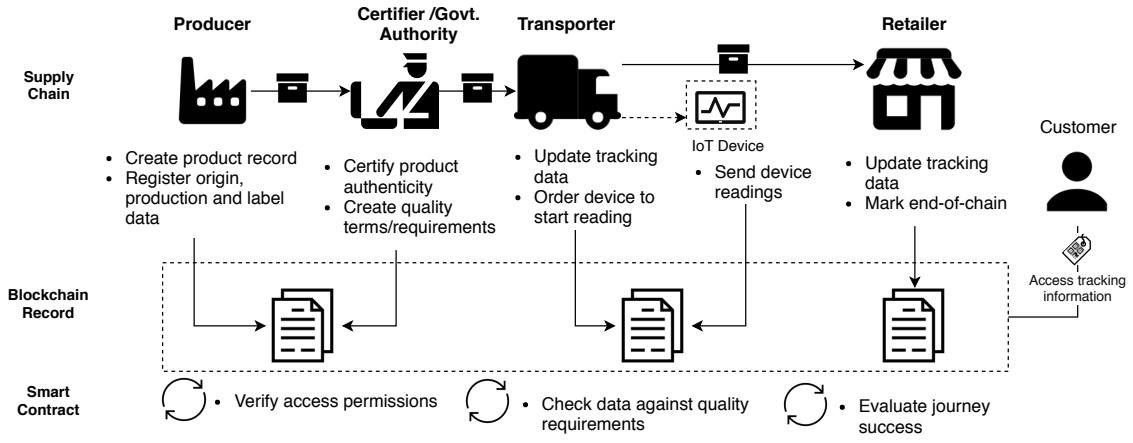


Figure 1.3: Conceptual view of supply chain tracking of goods supported by blockchain and smart contracts.

Entities that have access to the system (i.e.producers, transporters or certifiers) input into the blockchain smart contract system updates on status and location of the products they carry. Authentication is handled with smart contracts via a RBAC (Role-Based Access Control) system, in which permissions for interacting with the system's smart contracts are dependent on what role or roles each user has. These are automatically verified with each user's access attempt.

Another goal is to automatize or improve some of the steps of tracking and monitoring products by connecting IoT devices to the blockchain. Using sensors and other kinds of tracking devices, it is possible to measure and afterwards register on the blockchain information on the items condition, location or status. For this purpose, a system for users to control their devices via orders sent through our blockchain system is implemented.

An additional feature that is explored is a system for enforcing customizable quality standards, where all of the information introduced into the blockchain - whether by devices or by authorized users - is continuously tested against any terms for transport, quality or clearance that authorities and certifiers might have seen fit to enforce. A feature like this seeks to ensure buyers, or other parties that have stake in the product, of the integrity of the items being supplied.

The intent behind having all of the functionalities described above working in tandem is to achieve a blockchain-based application for supply chain tracking that is more transparent and trustworthy than the ones currently in place, tackling some of the issues

in logistics that were previously described.

It must be pointed out that this solution does not contemplate the use of cryptocurrencies or payment features in the system. Although these can certainly be very valuable tools, for now the focus of the system is on blockchain's use as a decentralized, untamperable registry for logistics tracing, leaving the financial aspects behind these processes for future work.

For the system implementation, Ethereum smart contract technologies will be used. Ethereum is the platform that kickstarted smart contract technologies, and is overall one of the better documented and more used frameworks for development and test of complex blockchain applications. After implementation, we will make tests and simulations of real-life supply chain processes being tracked with our platform. A final evaluation of the results obtained, including challenges and future prospects for blockchain in this field will be made.

The conclusions taken will hopefully contribute to the general scientific study of blockchain and particularly its implementation and use in the logistics industry, as well as with IoT technologies.

1.2.3 Objectives and Contributions

The general objectives for this dissertation's work can be summarized as follows:

1. With blockchain technologies, design a smart contract system that can be used by different entities to record and to access different products data as they travel through supply chains. The system should take advantage of blockchain technologies main benefits - trust, security, transparency and untamperability;
2. To automate some of the validation, certification and quality checks that happen in supply chains by taking advantage of smart contract technologies' capabilities for complex logic processing in blockchain systems;
3. To study and implement IoT device communciation with a blockchain system;
4. To find the most adequate development tools available to complete the first three objectives, and find out what benefits and drawbacks might be expected from using them.

And along the work developed in the dissertation, the following contributions have been made:

- The creation of an Ethereum smart contract system, developed in Solidity (a smart contract coding language) using Truffle Suite tools. The system fulfils objectives 1-3.
- The development of two browser-based APIs to interact with the smart contract system:

  - The first application accesses the blockchain system to show the logged in user the tracking data of the items in his possession. It was built in Javascript and HTML, and uses many different tools and libraries, which are detailed in Section 4.3.
  - The second application focuses on bridging the communication gap between an IoT device and the blockchain-based smart contract system. To implement it, the same tools as the first application were used plus an Arduino UNO microcontroller connected to a temperature sensor. From it, measurements are taken, and registered on the blockchain upon the receiving of temperature reading requests.

- An evaluation of the implementation process and of the final solution's positive and negative points. A number of conclusions are reached on possible points for improvement and issues found that gives some insight on blockchain technology's readiness and fitness for being adopted in logistics systems.

1.3 Document Structure

This section provides a summary of the following chapters of this dissertation.

- Chapter 2 covers the state of the art, beginning with an overview of blockchain technology's most basic mechanisms and some of the existing frameworks for application development. This is followed by a section detailing our chosen framework, Ethereum, and the intricacies of its smart contract technology. The final section of this chapter is an overview of the research that has been made on the fields of blockchain for logistics and blockchain being used with IoT technologies, both by academic researchers and by companies.
- Chapter 3 gives a detailed explanation of our system model. It begins with an application scenario, followed by sections detailing the system's objectives and functionalities, the overall design and the challenges faced when developing our solution.
- Chapter 4 discusses how we implemented our system model, what tools were used, the final structure of our smart contract system and how it operates. We also discuss the implementation of two web-based applications we built that interact with the blockchain - one for users and one for bridging communication with an IoT device.
- Chapter 5 details the results of our work - the smart contract system and its applications. We evaluate the system's overall performance, cost and scalability.
- Chapter 6 contains conclusions on the results and work of this dissertation, final remarks and a proposal for future work.

Chapter 2

State of the Art

The state of the art chapter will give an introductory overlook of the important concepts behind the work of this thesis. The chapter starts with a section explain blockchain technology in its simplest form (the bitcoin model) and its recent innovations, followed by a section explaining important concepts specific to Ethereum technologies and finally a section analysing the more recent works and applications on the fields of blockchain for logistics, smart contracts and IoT, and a short reflection on the research done.

2.1 Blockchain Technology

One of the main objectives of this thesis is studying the potential benefits that the blockchain technology could bring to logistics. Blockchain is a DLT (Decentralized Ledger Technology), first created and used by Satoshi Nakamoto in the Bitcoin cryptocurrency transaction system [3]. Its functioning has been summarily explained in Section1.1.1, but a full and detailed explanation of the technologies that have made blockchain possible will be given in this section - starting with an explanation of some important concepts behind the definition of blockchain.

Blockchain is a DLT where a data ledger in the form of a chronological chain of blocks is constructed in a P2P network - hence the name blockchain. It was first created and used by Satoshi Nakamoto in the bitcoin cryptocurrency transaction system [3].

There are a few important components to be found in a blockchain:

1. blocks are data structures containing the information being stored in the blockchain. When they are added to the blockchain, they are immutable and irreversible.

2. consensus is the system by which users participating in the blockchain network come to agreement on a block's validity and if it can be added to the existing chain.

Bitcoin in particular uses the proof-of-work system.

3. nodes are users of the blockchain network. They participate in the consensus system and in the process of validating blocks by providing computing power.

Cryptocurrencies use blockchain to make secure transactions of digital coins, but this is not the only possible use for this technology. It can be used for purposes of validation, tracking, registry, or any other task that requires secure and immutable data storage and/or digital transactions. However, since the bitcoin blockchain is the most traditional and well-known use of blockchain, it will be the base example used throughout Chapter 2 to explain the most basic technical details of blockchain technology.

In the bitcoin blockchain many technologies are used in tandem, namely timestamping of transactions, peer-to-peer networks, cryptography, and shared computational power via a consensus algorithm called proof-of-work.

2.1.1 Data Encryption

To understand how blocks, mining and consensus ensure that the blockchain is secure and immutable, one needs to understand the how the core encryption of blocks is made - which is mainly through the use of **hash functions**.

A hash function can be generally described by Equation 2.1:

$$hash(s) \rightarrow p \tag{2.1}$$

Both s and p being strings. The definition of hash function is a one-way, non invertible encryption function that maps a set of inputs to a set of outputs. This means that $hash^{-1}(p) \rightarrow s$ does not exist. A hashing function is deterministc, in the sense that the same input s always returns the same output p. Another important characteristic is that a small change in the hashing function's input produces a very different result in the output [18].

The bitcoin system uses a hashing function called SHA-256, where the output string p always has a fixed length of 32 bytes [18]. New blocks must always contain the hash (output p) of the previous block in the blockchain, so changing any block would require recalculating the hashes on subsequent blocks as well [19]. To further increase the security of the blockchain, a Merkle Tree system, involving more complex hashing, is also used.

It is important to note that, as shown in Figure 2.1 blocks in a blockchain are linked through each others hashes, one of the reasons why it is almost impossible to cheat the information on them. The linking of blocks through hashes, and the inclusion of timestamps and a Merkle Roots inside them are some of the most important mechanisms that secure the blockchain and make it so difficult to tamper with.

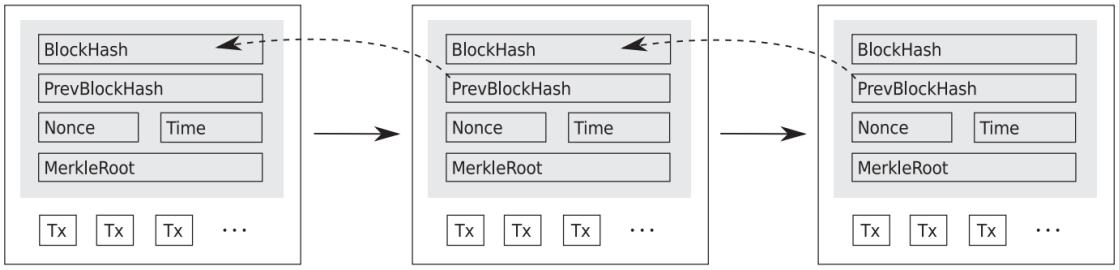


Figure 2.1: Simplified representation of consecutive block are linking through each other's hashes. From [21].

2.1.2 Blocks

The pieces that constitute the blockchain are called blocks. Blocks are up to 1MB in size, most of which is transaction data - however, the most important part of the block is the 80 bytes forming the **block header**. The block header contains data that is used during the mining of blocks and to ensure the overall security of the blockchain. It is composed of five pieces of data [19], and two of them are hashes (see Section 2.1.1), namely:

- **The Current Block's Hash** - 32 byte string, obtained by running the SHA-256 hashing function on a string representation of the current block.
- **The Previous Block's Hash** - hash of the block that precedes the current one.
- **The Merkle Root** - Also called binary hash root or Root Hash, it is the result of applying the Merkle Tree algorithm to every transaction in the block, also with the SHA-256 hashing function.

The other three pieces of data are not hashed, but are important information related to the process of block mining. They can be summarily explained:

- **The Timestamp** - A 4 byte timestamp encoded in Unix "Epoch"format. Blocks don't need to be in chronological order within the blockchain, but to be accepted, their timestamp must be greater than the median timestamp of the last eleven blocks and lesser than the median timestamp of all nodes connected to the miner + 2h. Block timestamps are overall only accurate within one or two hours, but they exist to make the block more difficult to hack [22].
- **The Nonce** - A 4 byte numeric counter, incremented every time an attempt to solve the proof-of-work problem within the set difficulty target is made during mining. When the answer to the problem is found and a new block can be made, the counter can stop being incremented.
- **The Difficulty Target** - A 4 byte numeric representation of the accepted difficulty for mining. Although it is not represented in 2.1, it is important to the process of PoW consensus, explained in Section 2.1.3 [22].

2.1.3 Proof-of-Work

To make sure that creating fake blocks requires more investment than potential returns, the blockchain network demands proof-of-work (PoW), which is the mathematically complex problem miners must solve when trying to create a new blocks. There are several different algorithms for this in existence (Ethereum, for example, has its own proof-of-work algorithm called Ethash[23]), but the most well known is the one used in bitcoin.

In the bitcoin blockchain, proof-of-work is finding a **block hash** with a set number of leading zeroes - defined by the difficulty target parameter on the block header. This can be done by running the SHA-256 hashing function on the block data with a slightly different input every time. The varying input is called **nonce**, also present on the block header, and usually some sort of incremented counter. Figure 2.2 shows the inputs used on the SHA-256 when trying to find a block hash with a certain level of difficulty.

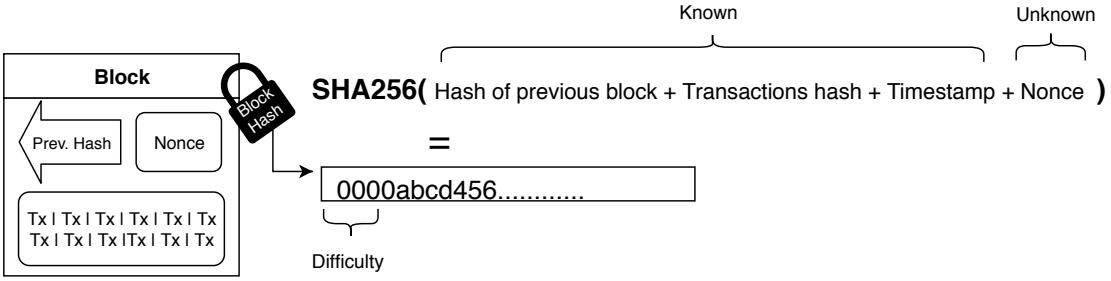


Figure 2.2: Graphic illustrating the expected result from the proof-of-work method using the SHA-256 hash function.

2.1.4 Transactions

Bitcoin transactions are fairly simple in functioning. The user making the payment - the sender - creates a transaction, which has inputs and outputs, as seen on Figure 2.3 on the block showing transaction data. Inputs are references to previous transactions where the sender obtained bitcoin. This bitcoin is sent to another user - the receiver - in an established amount making this person the first output's recipient. If there is any spare change from the input transactions to the first output, it is transferred back to the sender, who is the second output's recipient [21] [3].

The security of transactions is guaranteed through a key pair signing system. Each user has two keys - a public key and a private key, mathematically related via an asymmetric elliptical curve function. As the names suggest, only the user knows his own private key, while his public key is shared when he is making transactions. The keys are mathematically related, but due to the intractable property of elliptic curve cryptography, the private key is almost impossible to find with just the public key.

As illustrated in Figure 2.3, when the sender is making a transaction, he creates a signature for the transaction with his private key. This is done using a specific signing function with three inputs - the sender's private key, the receiver's public key and the

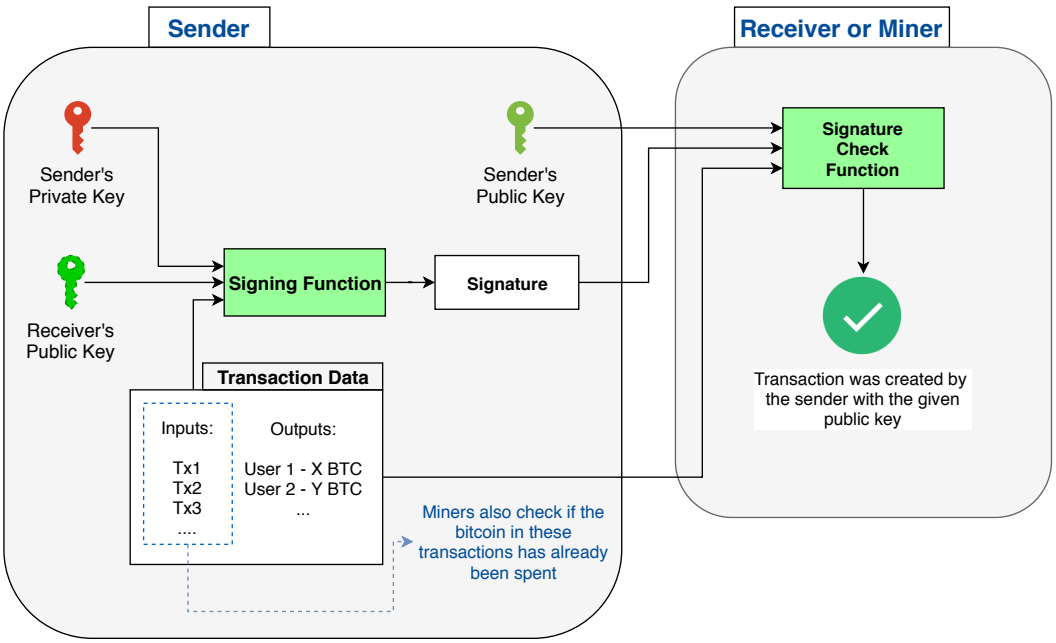


Figure 2.3: Diagram illustrating the process of creating, signing and validating a transaction using a key pair encryption system.

transaction data. The sender's public key is known, and can be used, along with the signature and the transaction data, to verify that the transaction has indeed been created by the person with that public key. Miners also do signature checks when adding transactions to a block.

To avoid the case where users to spend the same coin twice - double spending - bitcoin has a network-wide register of unspent transaction outputs called UTXO pool [21]. If the bitcoin the sender references as his inputs are in this register, he hasn't spent it yet, and can do so in a new transaction. Miners do this check for every transaction input, as well as checks for repeated inputs (double spending in the same transaction).

2.1.5 Mining

Mining and consensus are the systems bitcoin has in place that enable its functioning as a secure, distributed and permissionless cryptocurrency network. As was summarized in Section 1.1.1, the bitcoin network is comprised of a network of users called nodes. All nodes contain a copy of the blockchain ledger, which is a register of the bitcoin transactions that occur between users. Nodes that validate transactions and compete to create blocks are called miners. The process for verifying and creating a block involves the solving of a mathematical problem that demands a lot of processing power, as well as validation by other nodes. The consensus system put in place to create blocks and reach an agreement on new block's validity is called proof-of-work. If their block is chosen to be added to the blockchain, miners are rewarded for their work in maintaining the network secure with bitcoin. The generic description of the process is depicted in Figure 2.4.

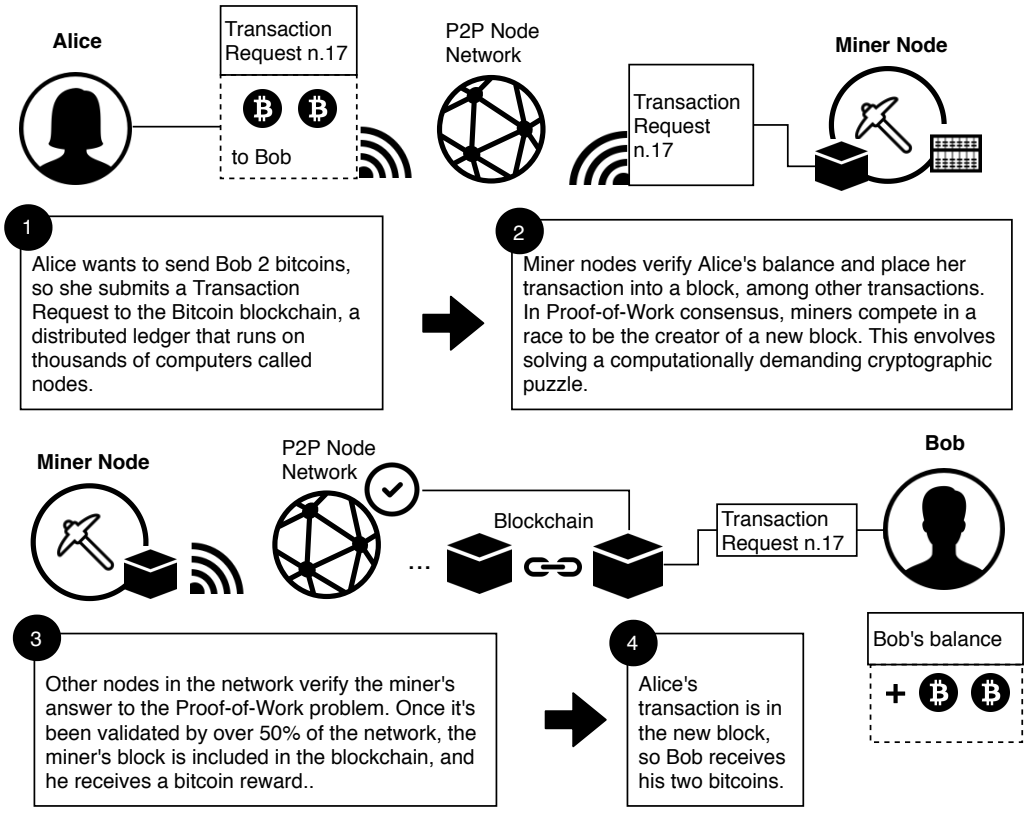


Figure 2.4: Diagram of the verification process of a bitcoin transaction between Alice and Bob.

One of the problems that can occur is forking, where two valid blocks are created nearly at the same time and lead to a fork (divergence) in the blockchain. However, as miners continue building the chain, at some point one of the forked block branches becomes larger than the other, and the smaller one is rendered invalid and obsolete [24], so while long forks are possible they are unlikely. Newer blockchains try to address this issue and provide finality, which is the absence of forks in the system.

The bitcoin network adjusts its difficulty level with every 2016 blocks created, so that the average time for mining a new block is always approximately 10 minutes [22]. As the bitcoin price goes up, so does the difficulty level for mining. For this reason an increasing number of energetic resources is spent by competing miners as the processing power demanded to hit the difficulty target goes up. If their block is chosen, miners are currently awarded 12.5 BTC on the first transaction of the block. While this serves as incentive for miners to keep working on maintaining the network, the difficulty level can only be maintained as long as the reward for mining is greater than the price of energy spent.

2.1.6 Challenges for developers

There are a number of problems inherent to blockchain technologies that make the development of new applications difficult, namely:

1. **Blockchain does not scale well** - Scalability is always a problem for blockchain systems. As they start becoming too big, their upkeep is progressively more demanding in terms of computing power and storage [13].

2. **Blockchain systems are expensive and slow** - Because validating every transaction requires all nodes in the system to verify it, there is a lot of computing power needed to run a blockchain system, and this often translates into higher costs and latency than what is found in traditional centralized systems.

3. **There is no privacy** - Even if a blockchain system is permissioned, all nodes possess the block data that can be decrypted with little effort unless the system has been designed with privacy concerns in mind (i.e. Quorum[25] or Hyperledger Fabric[26]). Therefore, maintaining confidentiality between nodes is difficult. This can be a downside to many applications, including the one developed in this dissertation's work.

4. **Blockchain is a very recent technology** - a lot of the hype centered around blockchain systems is speculation. While the technology has a lot of verified strong points, its usability for applications that aren't cryptocurrencies has yet to be completely explored, developed and improved.

The problems enumerated are largely dependant on the improvement of development tools and on the tweaking of some of the processes behind blockchain. The large array of frameworks being developed tackle these issues on varying levels, but generally tradeoffs must be made, since many of these solutions compromise other positive aspects of blockchain, like security, transparency or immutability.

2.1.7 Frameworks

The growing excitement surrounding blockchain technology's potential has led to the creation of many different frameworks for developers to work on. Some are geared towards particular uses - private ledgers, banking operations, asset tracking - while others are more versatile and allow the development of many different systems.

Dinh et al. in their paper "Untangling Blockchain: A Data Processing View"[27] provide a very complete overview of the existing frameworks and their distinguishing features. Of the many options available, the ones that stand out the most are Hyperledger and Ethereum - Hyperledger because it is designed towards use in consortium blockchains (blockchains whose users encompass several different companies), and Ethereum because of its smart contract technology, which is at the forefront in terms of versatility.

Hyperledger : a consortium of open source projects hosted by the Linux Foundation since 2015. Its purpose is mainly to create a bridge between leader companies in different areas - such as banking, finance, technology, IoT, manufacturing - in the joint advancement of blockchain technologies.

Hyperledger is a very popular platform and has produced several different frameworks for developers to work with, most notably:

1. Hyperledger Sawtooth - a framework for building DLTs geared towards enterprise or permissioned use, its main focuses are modularity and extensibility. Sawtooth technology is designed to support several different smart contract languages (including Ethereum's Solidity), and also features built-in access control and role attribution. Sawtooth operates PoEt consensus, but minor adjustments to this mechanism are allowed. The project's future goals include developing better privacy settings, which are not existent at the time [28].

2. Hyperledger Fabric - developed with the support of IBM, Hyperledger Fabric's main feature is its modular architecture, which separates transaction ordering from chaincode execution - meaning that there can exist a much greater privacy level between nodes in Fabric blockchains. Privacy is a very valued characteristic by companies and enterprises and one of the main drawbacks of traditional blockchain technology. Due to its modular nature, Fabric can also support many different consensus systems. It supports permissioned blockchains and smart contracts written in Go [26]. Recently it has developed support for the EVM as well.

3. Hyperledger Burrow - a permissioned blockchain node that runs the EVM. Worthy of note is that the original Ethereum network does not support permissioned blockchains. Burrow uses a PoS consensus system called Tendermint, and provides high transaction-throughput and finality [29].

Hyperledger was designed with use for consortium chains in mind, and that certainly fits the scenario that this dissertation is looking to achieve. It allows for better throughput of transactions and less latency than Ethereum. However it suffers from scalability issues[27].

Ethereum : a blockchain network proposed in 2014 by Vitalin Buterik, a former Bitcoin programmer, Ethereum is nowadays both the second largest cryptocurrency network in the market and a popular framework for developing and testing blockchains. It was a pioneering framework in the field of smart contract technology, introducing an Ethereum Virtual Machine (EVM) that runs smart contracts written in Solidity, a Turing complete language [23]. Users test contracts on the EVM before launching them on the network for a small fee called 'gas', dependent on the size of the smart contract instructions. Furthermore, Ethereum posesses a coin called Ether that can be used to power applications on

the Ethereum blockchain, commonly referred to as Dapps (more information on this is found on Section2.2.1).

As a framework, Ethereum is very popular. It is a pioneer in the development of blockchain and smart contract technologies, and has proved immensely popular, with an enthusiastic community and solid documentation to back it. The technology is so popular that other frameworks have implemented support for Solidity smart contracts and EVMs on their platforms (ex. Hyperledger Burrow, Hyperledger Sawtooth). Others like Quorum and Parity utilize the EVM as their smart contract execution environment [27].

However the main Ethereum blockchain does not directly support features like privacy, permissioned networks or other consensus systems besides its own (a PoW based system named Ethash), mainly promoting the creation and use of public applications (Dapps). This does not mean there are no other options to be taken on this matter. Open-source code is made available to developers who want to create their own private Ethereum blockchains [30], and projects such as Quorum[25], which uses the EVM, tackle the privacy issues of blockchains as well. So using existing Ethereum development tools to obtain a system with permissioned use in mind is certainly valid, and as the technology advances it will become easier to put in practice.

2.2 Ethereum Blockchain Technology

Ethereum blockchain technologies were chosen for the implementation of the blockchain-based logistics tracking system proposed on this thesis. This decision came down to a few reasons that are explained in Section 4.1.

As such, this section focuses on giving an overview of important, specific concepts relating to Ethereum blockchain, that were not previously covered on Section 2.1.7.

2.2.1 Ethereum Smart Contracts

Smart contracts are a type of transactional technology first proposed by Nick Szabo in 1997, as a form of controlled and secure digital contract that can be enforced and embedded onto property. They would be essentially security protocols, protecting and ensuring ownership within contracted terms [31].

In 2014 Vitalin Buterik proposed a new form of blockchain system called Ethereum, that sought to solve the Bitcoin blockchain's lack of versatility for scripting in transactions. His system introduced a blockchain with a built-in Turing-complete programming language, which can let anyone write smart contracts and decentralized applications. In this blockchain, users can create their own arbitrary rules for ownership, transaction formats and state transition functions [23]. With Ethereum, Nick Szabo's concept of smart contracts found its existence within the decentralized blockchain system [9].

For smart contracts to be embedded into blockchain systems, their design and functioning has to undergo some big changes. Ethereum differs from traditional blockchain because it is a transaction-based state machine. There exists a genesis state, and through the execution of transactions the Ethereum blockchain's state is morphed, and can be validated from the information present in the latest block accepted to the network. The current state is stored in the state database, which is kept off-chain and has a complicated structure made-up of mappings or hash tables.

Ethereum's state is made up of objects called accounts[23].There are two types of account in Ethereum: externally owned accounts controlled by private keys (corresponding to user accounts), and contract accounts, which are controlled by their contract code. This last type is what we commonly call smart contracts. All accounts contain a nonce (a kind of transaction counter) and a balance, and the ability to sign and send messages/transactions. Smart contract accounts are also associated to two additional data fields - one for contract code and another for contract data/state storage[8].

Transactions in Ethereum pertain to not only the transaction of cryptocurrency, but also the execution of any smart contract code that alters the state of the Ethereum blockchain. Figure 2.5 illustrates an example usage of a smart contract that implements an to-do list, owned by Bob but potentially usable by Alice after he updates the contract's state. It shows also how state variables are used to control the smart contract's behaviour, which functions as a state machine run on the blockchain.

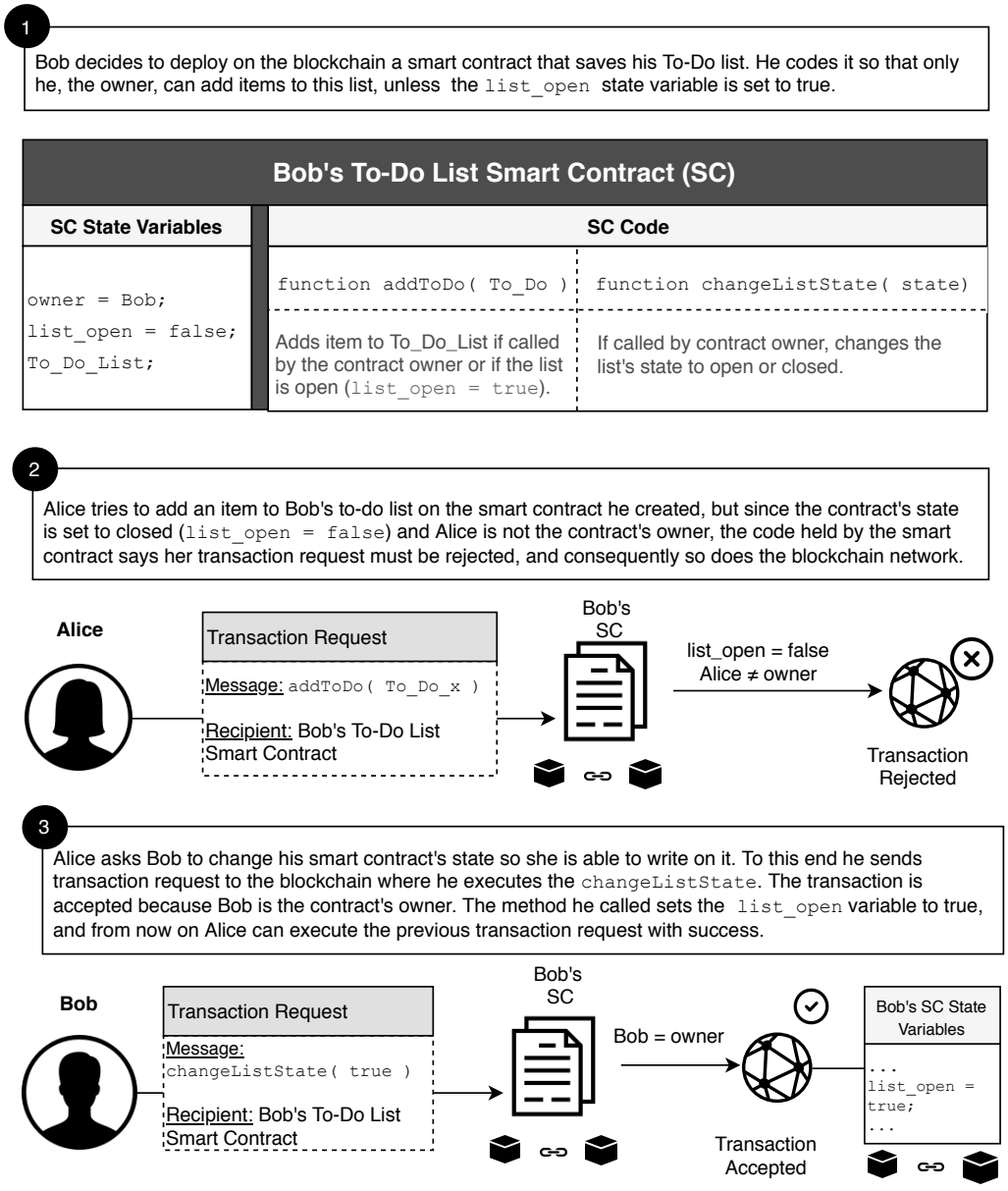


Figure 2.5: Diagram of the verification process of a smart contract (SC) transaction between Alice and Bob.

When creating a contract, developers define its behaviour through code, which is unmodifiable after being launched in the blockchain. Contract state changes can occur for transacting a coin or token - which contract accounts, like user accounts, can hold - or for updating state variables inside that contract, which is what Bob does in Figure 2.5. State variables can be used to store data, but in the Ethereum blockchain this practice is avoided due to storage being expensive.

In terms of the blockchain technology, smart contracts had to come with several innovations. Although the overall mining process is similar to Bitcoin's, Ethereum developed its own PoW consensus system called Ethash and a new form of Merkle Tree called Patricia Tree [23]. While bitcoin blocks store information on transactions only, Ethereum stores information of both transactions and the most recent state of every contract and user account.

As they exist nowadays, smart contracts are protocols that verify, secure and enact transactions or agreements between consenting parties in a decentralized network. They are one of the most powerful assets supporting the rise of blockchain technology, and are already used to power many applications in areas such as governance, crodwfunding, autonomous banks, keyless access and others, including logistics [9].

2.2.2 The Ethereum Virtual Machine (EVM)

Smart contract code itself is executed by the Ethereum Virtual Machine or EVM in the form of a low-level, stack-based bytecode language, referred to as EVM code, and made up of series of bytes where each byte represents an operation.

The EVM is a stack machine, and uses a last-in-first-out stack container to push and pop values as it operates, as well as an expandable memory byte array and, additionally, the smart contract's long-term storage. Of these three types of storage, only contract storage remains after code execution. At each point of execution, a data packet called tuple contains the computational state of the EVM at the given moment, and is used throughout the iteration of instructions from the beginning to the end of the code being processed [23].

The execution model of the EVM has been described as quite simple, and its overall lack of efficiency is one of the points for improvement that developers have identified. Even executing basic operations to the functioning of Ethereum, such as signature verification, updating the Merkle tree and state databases is very computationally expensive[30]. While the current platform is admittedly inefficient and inappropriate for complex applications, plans are being laid out for new versions of the EVM that address these issues[32].

EVM code can be written and compiled in a higher-level language, Solidity. Other Turing-complete languages are used to write smart contracts on other frameworks, such as Java or Go, used in Hyperledger Sawtooth and Fabric [28].

2.2.3 Solidity

Solidity is a high-level language used to create smart contracts that can be compiled into the EVM. While there exist other high-level languages for this purpose, Solidity is one of the most used, particularly it is the language used in development of this thesis proposed blockchain application. As such, some basic concepts particular to this language can be useful for understanding later chapters discussing implementation and results. The information given in this section is a condensed version of concepts explained in the Solidity Documentation pages[33]. To help visualize and understand them, i use an excerpt of the Ownable.sol smart contract code, taken from an open-source smart contract library and shown in Listing 2.1.

```
1  pragma solidity ^0.5.2;
2  /*** @title Ownable
3  * @dev The Ownable contract has an owner address, and provides basic
       authorization control functions */
4  contract Ownable {
5    address private _owner;
6
7    event OwnershipTransferred(address indexed previousOwner, address indexed
         newOwner);
8  /** @dev Throws if called by any account other than the owner.*/
9    modifier onlyOwner() {
10     require(isOwner());
11   _;
12   }
13 /** @return true if `msg.sender` is the owner of the contract.*/
14   function isOwner() public view returns (bool) {
15     return msg.sender == _owner;
16   }
17   function renounceOwnership() public onlyOwner {
18     emit OwnershipTransferred(_owner, address(0));
19   _ owner = address(0);
20   }
21 ...
22 }
```

Listing 2.1: Excerpt of the Ownable.sol smart contract code. It is taken from the OpenZeppelin repository[34].

Types Solidity supports basic variable types such as boolean, integers (signed and unsigned, up to 256 bytes), bytes and their arrays, as well as basic logic and number operations. Dynamic byte arrays can be declared as strings. New types can be defined in the form of `structs`, which are made up of basic type variables.
For storing data, users can utlize arrays or a type called `mapping`, which is essentially a kind of hash table where values are mapped to keys. Solidity does not support floating point type numbers in their entirety, yet, and has some limitations when it comes to dealing with dynamic arrays.
Solidity also has a variable type that is particular to blockchains and the EVM, called `address`. An Ethereum address is a 20 byte value that points to where a user or contract account is stored in the blockchain. The address type serves to store these values when needed. Addresses also have members specific to their type, mostly used when transferring Ether.

Contracts Contracts are akin to classes in other object-oriented programming languages. As can be observed in the Ownable contract of Listing 2.1, they can contain state variables, methods, objects, and can inherit these attributes from other contracts as well. Users interact with contracts and alter their state by calling their functions.
There are also special types of contracts called libraries and interfaces. Ordinary contracts may use library methods to access external code that obtains/calculates values for them, but cannot directly alter the calling contract's state. Interfaces are used for bridging communication between two different contracts launched on the same blockchain.

State Variables State variables are kept in the contracts storage, and make up its state. They are the most expensive kind of storage in the blockchain (this is discussed in Section 2.2.5). They can also be declared with some of the visibility types explained in the functions section of this list.The Ownable contract example has only one state variable, the `_owner` address.

Global Variables Some variables pertaining to the block and transaction information are available to smart contracts. One example is the `msg.sender`, used in the `isOwner()` method of the Ownable contract to access the address of the user making the function call. Other variables are accessible this way.

Functions

Functions in Solidity take parameters as inputs, and can return an arbitrary number of values as output. When they are declared, a number of keywords - `private`, `public`, `internal`, `external`, `pure` or `view` - can be combined to define a function's visibility and access to memory. The first three keywords we listed can also be used when declaring state variables. They can be summarily described as follows:

- `internal` functions are accessible from inside the contract or its derived contracts own code only;
- `external` functions are only accessible by users or other contracts;
- `public` functions are available both internally and externally;
- `private` functions are accessible from inside the main contract only, and inaccessible to derived contracts or other accounts
- `pure` functions do not read or modify state
- `view` functions can read state but won't modify it

The Ownable contract contains two example functions. Both were declared as public, but one is of the `view` type, and doesn't alter state.

Modifiers

Function modifiers are essentially mini-functions that are only run before executing a normal function's code, and can in this way modify their behaviour. They generally enforce requisites for the user to get access to a function.
For example, in the Ownable contract, a modifier named `onlyOwner()` exists, and its purpose is to check if the user making a function call is the contract's owner. In the declaration of `renounceOwnership()`, we see the `onlyOwner()` modifier being used. This means that any user trying to renounce ownership is first verified as being the owner, via the code run in the `onlyOwner()` modifier.

Events Events are interfaces for the EVM's logging functionalities. They are inheritable members of contracts. When called, their input arguments are inserted on the block as log data. This means the data is saved on the blockchain, but outside the EVM's state, making events a kind of cheaper storage than state variables.
In the Ownable contract, an `OwnershipTransferred` event is logged every time a user calls the `renounceOwnership()` function. It is possible to have front-end applications listen for events being logged on new blocks, and act according to their log data. The drawbacks to events are that it's not possible to restrict access to emitting them, and that transaction logs might eventually get deleted if the block is very old.

2.2.4 The Cost of Ethereum Blockchains

The possibilities of smart contracts are almost limitless, and hampered only by one thing - computing power, the only resource expended in maintaining a blockchain. In Ethereum blockchains, this resource can become extremely expensive, mostly due to efficiency issues of the overall Ethereum blockchain's design and its EVM execution model.

The Ethereum blockchain quantifies the computing power being expended with a unit called 'gas', and there are two variables associated to every transaction that define its quantity and its cost - they are respectively named gas cost and gas price; two different concepts with similar names[8].

Every computer instruction made when running smart contract code has a cost in gas, defined in Ethereum's yellow paper[8]. Consequently, every transaction interacting with a smart contract has a gas cost, which is the sum of the cost of every computer instruction being run in the smart contract's code.

When submitting a transaction to the blockchain, users define what gas price they are willing to pay in Ether (ETH), Ethereum's cryptocurrency, for each gas unit, in order to have their transaction included in the next block. The amount of ETH they spend on a single transaction is therefore the gas cost of executing its code times the gas price they set out to pay. Converting ETH into regular currency, we can tell the cost of these transactions tends to be quite high.

To put it into perspective, storing a 32 byte word in the main Ethereum blockchain costs 20.000 gas[8]. As of writing this, the average gas price is 2.4 Gwei [35] or nanoEther, and Ether's unitary price is approximately 87$ [36] (cryptocurrency prices are subject to a lot of variation, but the market is on a particularly low point, having hit its max at 1386$ per ETH in January 2018). Knowing this, we can infer that storing 1GB of data in the Ethereum blockchain right now would cost approximately 130,500$, a price that is very hard to justify, even for decentralized storage.

2.2.5 Storage in Ethereum

Data storage is one of functionalities our blockchain system most relies on, so this section reflects on some problems and benefits to the two kinds of storage that can be used in Ethereum blockchains - contract storage (state variables) events and off-chain storage. The first two have already been mentioned and their mechanics described in Section 2.2.3.

State Variables State variables that are stored in smart contract storage and therefore the EVM. While smart contracts can have unlimited data storage, it is very expensive to write on, particularly because every blockchain node must have a copy of this data, which is kept in the state database, in their machine[8].

If we consider a permissioned blockchain, its users could decide to maintain the system and cover computational costs at their own expense. However, when no limits are imposed on blockchain operations, the problem of scalability quickly arises. If the network is large and too many users are storing large amounts of data on the blockchain, its state machine might become too big for nodes to hold in their computers. Additionally, if there is no monetary incentive to max out the computational power being provided to the blockchain (especially when using a PoW consensus system), the network becomes more susceptible to attacks, even if it is permissioned.

Events An alternative, cheaper storage method in Ethereum blockchain exists in the form of events.Events are basically a data packet that is written onto that block's transaction log, but not directly into the blockchain's state machine [33].

Consequently, not all nodes have to have the information of event logs saved on their computer, since generally block hashes suffice for consensus. Events will still be present in the blockchain, because each one is part of a block and therefore essential to validating its hash - but they are written in a kind of disposable log, and are consequently a much cheaper form of storage.

As the blockchain grows, accessing event data can become time consuming, especially if one isn't sure on what interval of blocks they should search for it. Another drawback is that access control cannot be enforced for emitting events, so the data being stored isn't verifiable as coming from a reliable source. Furthermore, as blocks age their log data loses relevance in validating the blockchain, and they might eventually get deleted.

Off-Chain Storage Another option Ethereum developers can take is using the blockchain to store validator hashes of databases and have the database itself exist outside the blockchain system. If users want to validate the trustworthiness of the data they are accessing, they can hash it and compare the result to the hash stored in the blockchain.

This kind of system can be very useful to save on the high costs of Ethereum state storage while taking advantage of blockchain's untamperability. The downside is that

the integrity of the original data set is not guaranteed the same way that state variables are due to being kept in a decentralized ledger. Data in external database systems is vulnerable to malicious attacks that can destroy or alter it, and just its hash is not enough to restore it.

So in conclusion, while smart contracts and Ethereum technologies in general allow developers a large degree of freedom, there are some inherent problems of blockchain in terms of price, scalability and efficiency, which are particularly significant when it comes to storage. In a private blockchain, costs and storage constraints could be managed and mitigated, but it is still a prohibitive factor in terms of the scale a decentralized blockchain system can have. As Ethereum technologies are updated and further developed, however, these problems can be improved on and perhaps eventually overcome.

2.3 Related Work

Most of the initial research done on blockchain technologies was derived from the work and study of independent developers, who saw potential in the technology powering Bitcoin.

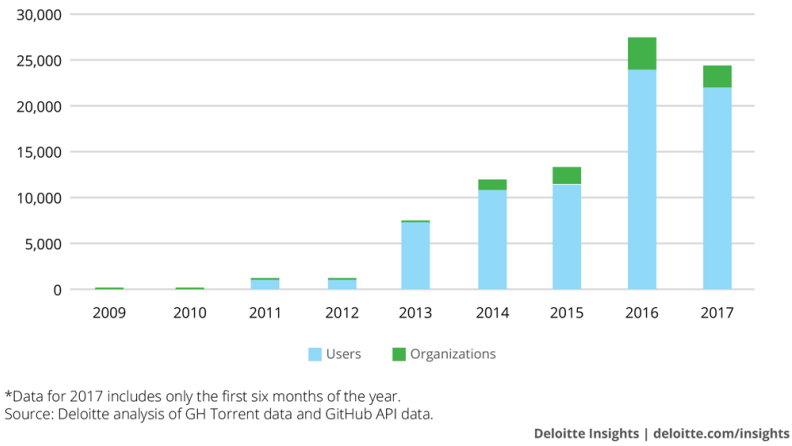


Figure 2.6: Number of new Github projects on blockchain, from 2009 to mid 2017, separated by author type. From [37].

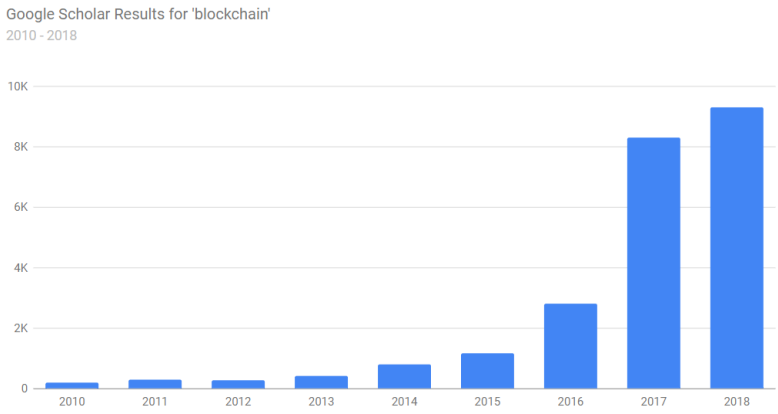


Figure 2.7: Number of Google Scholar search results on the topic of blockchain, from 2010 to August 2018. From [38]

Only since about 2013 have organizations, enterprises and start-ups picked up interest and started investing in this field [37]. This means that most research on blockchain is very new.

From Figure 2.6 we can see that independent developers are the main driving force behind blockchain projects on Github, being clear that only in recent years have the contributions made by organizations or companies reached significant numbers - approximately 10% of the almost 30.000 projects created in 2016, a number of projects that data suggests would have doubled for 2017. Comparatively, Figure 2.7 shows the number of publications to be found under a 'blockchain' search on Google Scholar for recent years.

It shows that the scientific community was slower to adopt interest in blockchain, and produced significant amounts of study on this issue only very recently.

In fact, during the research done on this State of the Art, it became clear that an active community of developers and some innovative companies are some of the best sources on this topic, and that the scientific studies made are somewhat lacking in comparison. Nevertheless, this section will attempt to cover some of the more recent developments on both the entrepreneurial and scientific side for blockchain adapted to logistics and IoT.

2.3.1 Research on Blockchain for Supply Chains and Logistics

2.3.1.1 Academic Studies

A number of studies have been addressing the use of blockchain and logistics, although many are sparse on technical details and few possess case studies or actual implementations of the system they propose. I will mention here those with the more signicant or innovative proposals to a blockchain and logistics system.

- In 2016, Yuan and Wang[39] proposed a blockchain driven intelligent transportation system, to their knowledge the first of its kind. It envisoned a seven layer system for transport-based applications, not limited to logistics; four of those layers described traditional blockchain system components (consensus, incentive, data and network), while the other were smart contracts, physical and application. The description of these last three layers suggested the use of smart contracts and some IoT devices for monitoring physical assets involved in transportation (vehicles, products, etc.) in a decentralized autonomous system. The applications could involve logistics or other transportation based systems, with the authors having chosen a ride-sharing system as their case study.

- A public supply chain management system using blockchain named CoC[40] is suggested by Xu et al in a 2017 paper. Users are separated into three groups, being particularly differentiated by the right to build blocks (public) and the right to submit records (logistics participants), a system the authors defend provides a hybrid form of DLT. Blocks are generated at the rhythm of a supply-demand system using PoW consensus, and encryption of records would be used to keep important information confidential. Technical details on some features are a bit sparse, and the overall proposition seems to rely heavily on technology that has not been fully developed.

- Feng Tian[41] proposed in his 2017 paper a blockchain RFID based traceability system for agri-food in a supply chain, designed specifically with food safety in Chinese markets in mind. It suggested the use of sensor-RFID to monitor products during transport, with a blockchain system guaranteeing that traceability of information

is reliable and authentic. RFID tags would be used in tagging, with other RFID-reading equipment being used in warehouses for inventory management purposes. While the author does not directly suggest the use of IoT technologies, he does mention that sensors for conditions like temeperature and humidity management could be used. A system like this could be used to prevent food safety problems, guarantee freshness and provide traceability, among other benefits. The main drawback, recognized by the author himself, would be its high monetary cost due to the current prices of RFID equipment.

- Ruta et al[42] describe in their 2017 poster abstract a decentralized, collaborative system for supply chain object discovery with semantic-enhanced blockchain discovery. Objects are registered onto the network with a qualitative description as well as other relevant information, like location or expiration date. When a node wants to find a certain type of object that is within his operating range, he makes a query, propagated by other nodes, that eventually selects and finds the closer sets of objects matching his description. In this way, a semantic based metric is combined with geographical distance to make a unique algorithm for item searching in a blockchain. The researchers implemented and tested their method and concluded it was feasible for medium-sized networks, although it required scalability adjustment. The methods researched in this poster are quite interesting and could prove very useful in a logistics blockchain, being used for example to find solutions for stock shortages or to fix difficulties in supply.

2.3.1.2 Enterprises and Independent Developers

Companies, start-ups and independent developers are as of now the biggest drive force behind blockchain technologies. On the field of logistics applications, some big investments have already started happening. Particularly interesting are the ones that provide open-source material, being due to that a lot more helpful to other developers in the community.

- Two industry giants of logistics and technology, Maersk and IBM, have started a joint venture to create a blockchain system that allows end-to-end shipment tracking. Each stakeholder of the supply chain can visualize the real-time progress of goods throughout it as well as the documents and bills associated to these products, with sensor and IoT tracking information also being made available. [13] By August 2018 this project had been launched and named TradeLens [43], with claims that throughout its tests there were cases where supply chain routes had been made up to 40% faster than usual. TradeLens, which uses Hyperledger in tandem with IBM Cloud, can be considered a large-scale test of the ideas that this dissertation explores, albeit without any open-source factors and an implementation that uses a different framework.

- An open-source project named Hyperledger Sawtooth Supply Chain [44] exists on Github, sporting 45 contributors and more than 5000 commits as of August 2018. As the name suggests, it is a blockchain system built with Hyperledger Sawtooth and directed towards supply-chain tracking of products. For this end, the blockchain keeps Records, objects with variable properties that are associated to products, and has a system for Agents that can alter or view these records. Transaction or alteration of records occurs with Proposals. The system seems therefore to have a form of permissioned use for record keeping and tracking of products. A demo of Sawtooth Supply Chain at work named FishNet[45] was made, for example, and shows how the blockchain can theoretically be used for asset tracking such as fish, although some features like privacy, role-based access control and document storage/certification don't seem to be directly available.

- Provenance [16] was one of the first proposals made on use of blockchain technology for supply chain transparency, particularly for providing proof of origin by certifier authorities in a shared, decentralized database based on Ethereum. Customers would be able to access supply chain data in their phones via a QR code tag to know the exact origins and certifications of what they are buying. Although the project proposed on their whitepaper does not seem to be fully complete, they have developed a software and made available a demo for companies, as well as put together a number of case studies. Similar projects exist, such as OriginTrail, Sweetbridge, Blockfreight or Ledgit, [46] with the difference that these companies utilize a token system characteristic of cryptocurrency based blockchain systems. Another noteworthy project is Waltonchain[47], which also aims to implement IoT devices in a tracking supply chain system.

- Other projects, more specific to certain subsets of the supply chain business, have been developed. Everledger [13] is a blockchain for tracking and certifiying the provenance of diamonds. Companies like Yamaha Gold and Emergent Technologies are looking to make an equivalent blockchain for gold tracking [48]. Ez Lab has a project for a blockchain for agricultural products like wine [49]. In Github, an Ethereum based, open-source project for supply chain tracking of coffee is availabe [50].

2.3.2 Research on Blockchain coupled with IoT

On their 2013 paper discussing Internet-of-Things technology and its future, researchers gave the following definition for what IoT is:

> *"[IoT technology is the] interconnection of sensing and actuating devices providing the ability to share information across platforms through a unified framework, developing a common operating picture for enabling innovative applications. This is achieved by seamless ubiquitous sensing, data analytics and information representation with Cloud computing as the unifying framework"*
>
> *-J. Gubbi et al in [51]*

In fact, IoT platforms, meant for processing and putting to use different sensor and device data, are one of the rising trends identified in the 2017 Gartner Hype Cycle. [7] Some of the most studied applications for IoT include automation of smart production/-factories and monitored or enhanced transportation processes [51], both of which are important processes of supply chains and logistics and therefore directly relate to this dissertation's work.

But although they address similar issues, how can IoT technology be coupled with blockchain technology? And the simple answer is, with smart contracts.

Sensor usage in supply chains and logistics to monitor parcels is a concept that every year turns more real, especially with increasingly cheaper sensors being made available. However, collecting sensor data for record purposes only is merely part of IoT's purpose. After this stage is done, a smart system that acts according to the data that has been collected is required. In a blockchain, this system would be smart contracts, programmed to trigger certain contract clauses when trusted IoT devices give notice of specific events.

Also very interesting is the issue of security in access control of devices, which is one of the main issues that hampers the use of IoT in real life [51], and could potentially be fixed by using blockchain. If these devices were programmed to obtain access control information directly from a tamper-proof and secure blockchain, their hacking by malicious third parties would be much more difficult to execute, and problems such as theft of information, data manipulation or disablement of the device or network could be prevented. The problem with this concept right now is the high level of processing power demanded by blockchain software, and which most IoT devices do not have. However, both companies, developers and academics have started studying possible ways to integrate sensor data or access control functions of IoT in blockchain, the most important of which will be mentioned throughout this chapter.

2.3.2.1 Academic Studies

There exist a considerable amount of articles studying combinations of blockchain and IoT. Worthy of note is that many of them mention applications in supply chain and logistics, although they are not the main focus of the articles themselves. Some of the most interesting articles and ideas are mentioned in this section.

- Christidis and Devetsikiotis[52] authored a paper on the topic of 'Blockchain and Smart Contracts for the Internet of Things', studying the potential of using these technologies in tandem. They suggest several interesting applications, such as using blockchain to securely transfer firmware updates to IoT devices and creating a marketplace of services between them. More interestingly, the authors also suggest use in supply chain transactions, giving the example of using smart trackers (particularly, ones that use BLE and GSM technology) on transporters and the containers they carry to automatically detect and register on the blockcain the occurrance of a physical transaction. They suggest that the devices should connect to gateways that provide them with access to the blockchain. They identify some problems for these applications, such as low transaction throughoutput, a moderate lack of privacy, the lack of legal enforceability of smart contracts and the need for well designed smart contracts to guarantee security.

- Kshetri [53] presented a study exploring how blockchain could enhance security for IoT that contained several insights into how both technologies complement each other. Not only can blockchain-based access control security be used for restricting access to IoT devices, but the decentralized nature of the network brings several benefits as well, particularly in comparison to the more commonly used systems of cloud computing for inter-device communication. Decentralization could bring diminished costs, security, transparency and overall more network availability for IoT devices. A case is also made for use of IoT in supply chains with blockchain, for tracing, pinpointing of faulty parts, identifying users of vulnerable devices and for registering updates, patches or part replacements.

- In their 2018 article, Pustišek and Kos [54] set out compare and analyse three different architectural approaches for the design of front-end IoT device applications based on Ethereum blockchain. Front-end applications are needed for both users and devices to utilize/access the blockchain. The authors identified two ways that an IoT device can be included in the blockchain: it can have its own set of keys and use them to interact with the blockchain network (create/receive transactions), or it can be a passive user, accessing only events or data readings from the blockchain. They also considered two different types of architecture: a stand-alone node, where both the front end application and geth program (essentially the blockchain client) reside in the IoT device, and a remote geth client based architecture, where the IoT device posesses only front-end software and communicates wirelessly with a geth

client run on another machine. They concluded the later option was more feasible because very few IoT devices have the processing power to run geth. They also concluded that in this remote geth client architecture, storing access keys in the IoT device is overall safer than having the key used by the IoT device stored in the machine that runs the geth client.

2.3.2.2 Enterprises and Independent Developers

- Some enterprises and developers have created a foundation named Trusted IoT Alliance [55], with the purpose of supporting the creation of an open-source, secure, scalable, interoperable, and trusted IoT ecosystem using blockchain. They established a sort of standard for IoT device registry in the blockchain and have some open-source code for Hyperledger available in their Github repository.

- Project IOTA, an open-source protocol run by a foundation with the same name, is an interesting take on the IoT and blockchain issue. It does not use blockchain technology, but instead something inspired by it, a DLT called Tangle. In this system, nodes dont have to validate all blocks being submitted - instead, whenever they are submitting a transaction to the ledger, they must validate two random preceding transactions before their own is put on the review list. Eventually, if considered valid, the submitted transaction is included in the ledger as well. Another big difference from blockchain is that there are no transaction fees, and since validation is not computationally demanding as it is with PoW, IoT devices can participate easily. The downside of this form of DLT is that it is not very safe when the number of transactions occurring in the network is small and can be easily overturned with access to large amounts of computing power. As it exists, IOTA needs a kind of centralized authority named The Coordinator to stay secure, which undermines the decentralized nature and associated benefits of the network. As of August 2018 there is no framework for IOTA, but it is being developed.

- IBM has started integrating Watson IoT, a product that analyzes and manages IoT device data, with their blockchain platform, so that partners can securely share this information. This is a component of their project on blockchain and supply chain, already mentioned in Section2.3.1.2.

- Weeve is a platform for securely trading trustworthy IoT data in a blockchain in exchange for cryptocurrency, creating what the developers call in their whitepaper [56] an Economy of Things (EoT). Their approach is to creating a scalable and safe system that bridges the gap between IoT and blockchain at both the hardware and software level. They create their own protocols, system architecture and Weeve wallet to this effect.

2.3.3 Reflections on the State of Art

After the research done for this dissertation, it becomes evident that although many companies and developers have already invested quite a bit in blockchain for use in logistics, the approaches to this issue vary wildly, aren't readily available and don't seem ready for large-scale adoption.

As has been noted, the scientific community has not been at the forefront on blockchain technologies and its development, at least not at the level that companies and independent developers have. Scientific study on this matter also exists, but with few applications that have been implemented, tested and put to practice. This paradigm seems to be changing, and it is important that it does, particularly because the technology lacks cross-platform standardization and proper documentation, issues that some form of community-wide consensus and studies could potentially fix.

Furthermore, there is still a lot of development to be done on blockchain technologies, particularly on the topics of smart contracts and their use with IoT devices. Smart contract technology is recent and its adoption and development is still occurring, and due to that the inclusion of IoT data in the blockchain is also a work in progress.

In lieu of this, the dissertation will focus more on developing, implementing and testing a functional smart contract framework for use in logistics, and the inclusion of IoT device data in it. Hopefully the approach taken will bring good results, but also insight on how smart contract systems like this should be designed, if they can be a good option for logistics applications and how much more development can or should be made on this issue.

Chapter 3

System Model

In accordance with the descriptions laid out in Section 1.2, we set out to model a blockchain system meant for tracking and tracing of products as they travel through supply chains.

This Chapter provides a conceptual, high-level overview of the system developed in this dissertation's work. First an application scenario is described and the system's objectives defined. Then, the plans for each of the components of this system are laid out and described in more detail - namely RBAC, product tracking, quality checks, support for different labels and IoT device implementation/communication.

3.1 Application Scenario

To better understand our proposal of a blockchain solution for application in logistics - namely what objectives it must fulfill and what processes it will improve - we can envision a typical scenario of a product being processed as it travels through the supply chain.

The hypothetical product being tracked is a large wine crate. It was produced in a Portuguese region well-known for wine quality, and has received a European certification for Denomination of Protected Origin (DOP). However, the wine must be submitted to a quality check, so before having it sold the producer must transport the crate to a wine cooperative's headquarters, where its certifiers will evaluate the wine's quality and origins in accordance with the DOP standards.

So the producer of the wine prints out a label for the product, along with some signed paperwork detailing its origin and other production data. He/she pays a transporter company to take the product to the cooperative's headquarters. Here the certifiers make quality checks and create a paper quality certificate for that wine, which is attached to the crate, and perhaps a digital one, saved on the certifier's database.

The wine crate is then taken to a nearby warehouse, where it stays for a few weeks.

Eventually a second transporter company takes it to a retailer, where the wine will be sold.

If the retailer wants to determine the wine crate's origin and journey conditions, he will probably check the paper registers that came with it, and since the journey described here is very short, there were likely no problems like missing records, forgery, mistakes or improper transport/storage conditions. As the supply chain complexity increases, the likeliness of these issues occurring grows.

Even if some of this tracking data is inserted into a shared and centralized database, which is managed by a central authority, its servers might experience issues, data might maliciously get altered/deleted, or the system might get hacked. Most of the existing tracking systems are limited to the owner company's operations anyway, because due to lack of trust or competitiveness in the industry, companies do not tend to share their platforms.

Furthermore, there is no way for parties with stake in the product to be informed of its journey from a decentralized source; if it has been completed successfully and within the standards they require. The producer wanted the wine to receive a quality check, but must communicate with the cooperative directly to find out if it did, or wait to receive this notification from the transporter. The cooperative's certifier passed the wine on its inspection, but wants to ensure it is stored at appropriate temperatures in order for it to not be spoiled.

Our proposal of a shared blockchain-based tracking system envisions to fix or improve some of these issues, by providing a common platform, that due to its decentralized nature, the intervening parties are more likely to trust. With it, the example scenario that was just described would instead unfold as is illustrated in Figure 3.1.

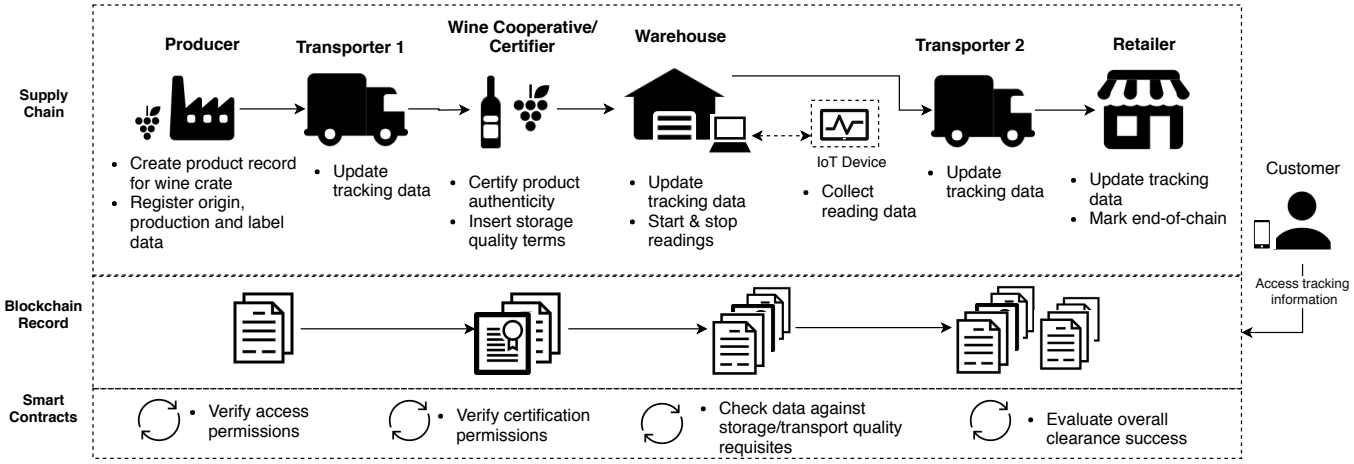


Figure 3.1: Representation of the tracking process of a wine crate scenario, using our proposed blockchain and smart contract technology solution.

As hypothesized before, a producer has a wine crate ready for shipment. Using credentials he obtained from a trusted regulator, he creates on our blockchain application

a new product record, which includes the wine crate's location, state, and some chosen label data. The producer intends for the wine crate to reach a certifier before being sold, so he creates two terms of quality control for the product's journey. It must pass by the certifiers location, and it must acquire the DOP type certification, both within a time limit.

When the transporter takes the wine crate, the producer registers the change of hands on the blockchain, and alters the product's state to 'in transit'. The transporter becomes the wine crate's carrier/bearer. Along the journey, he updates its location on the system, until he arrives at the intended wine cooperative's headquarters. As he logs this location, the coordinates match one the of quality control requisites set out by the producer, and happen to be made on time as well.

The transporter transfers possession of the wine crate to the cooperative, which becomes the crate's new bearer. As certifier authorities, the cooperative's workers analyse the wine product and record on the blockchain some state changes, followed by the attribution of the DOP certification, which can be accessed on the blockchain system as well. This fulfils the second quality control requisite created by the producer.

The certifier authority now intends to take the wine crate to a nearby warehouse. Before that, however, he creates on the blockchain a new quality control term, which institutes that the crate must be stored at an average 10 to 15 degrees Celsius, for example. The certifier then records the physical transfer of the crate to the warehouse manager, who becomes its bearer.

The wine crate's entry in the warehouse is registered through a label reader, and a computer application connected to both this reader and some sensors. It uses the label key to query the blockchain for any active quality terms on this item that it can fulfill. Finding the quality requirement for temperature, it starts registering the average temperatures in the warehouse until the label reader registers the wine crate's exit. When this occurs, the temperature reading is sent to the blockchain and validated against the rule set by the certifier.

After passing through a second transporter's hands and having its location and state updated, the crate reaches a retailer, who can access the blockchain to verify its label data, origin, travel path, bearers, certifications and even some storage conditions. He registers every individual wine bottle's label number in the blockchain, and has them point to the origin crate's record, so that via an application, costumers can use their smartphones to access this same tracking data and verify the wine's origin, as well as certain proof of its DOP certification.

Applied to a wide range of products, a blockchain-based application to track logistics operations could provide the basis for a platform that all these intervenients can trust and use. Smart contract technology can provide the validation tools and structures needed to store and process data.

Of course, a system like this is not completely foolproof. There is still a chance one of the intervening parties introduces wrong data or simply fails to introduce it, which

is why in this dissertation the integration of IoT solutions to automate some of these processes is taken into account. If a problem occurs in an item's transport, the existence of a trustworthy record where all actors have verified credentials, could help pinpoint the source of the issue.

3.1.1 System Objectives

With the application scenario described in Section 3.1 in mind, the objectives for our blockchain-based, logistics tracking application can be laid out. They are as follows:

1. The blockchain smart contract system will be permissioned (meaning only accessible by authorized and registered parties), and there will be an authentication system to distinguish between different entities and their respective permissions to act in the system. Access to any specific product tracking record and its update will be limited to the entity in possession of it. With these functionalities in place, the platform can avoid tampering and create trust between its users.

2. The system permits the tracking and tracing of items, with three mains status changes - location, current bearer and state - being registered in a product record that is kept on the blockchain, as well as other types of readings captured from IoT sensor devices. These were decided to be the most relevant and important kinds of tracking data to form good basis for transparency.

3. Certification given out by authorities can be registered. Quality checks or requirements for transport/storage can be defined by either these authorities or the producer, and automatically checked/enforced every time new data is input. These features exist to take advantage of smart contract's data evaluation capabilities, attempting to make certification more automatized but also more transparent and accessible.

4. IoT devices can communicate with the blockchain system and register any contextual data they collected on specific items. Their integration with blockchain systems is one of this dissertation's areas of study.

5. The system supports the storage of different formats or label types and standards. Label numbers can be used to access a product's record. This feature tests smart contract's modularity capabilities while seeking to support different logistics standards used for different products and their varying label formats.

Throughout the process of designing the system, these were the main objectives that this work set out to achieve.

3.2 General System Structure

Considering the objectives and functionalities intended for the system that are enumerated in the Section 3.1.1, it can be noticed that the first objective - a Role-Based Access Control - is needed to manage entities and their permissions, while the last four objectives/functionalities - product tracking, quality control/clearance, IoT device monitoring and support for multiple labels - pertain to managing and processing specific product units.

It is in fact fairly obvious that the object at the center of a supply chain process is the product and what we intend to create is its digital record. This digital record was simply named Product Record, and the smart contract system will be able to hold an indeterminate number of them. In this system a Product Record can represent a state database on all kinds of goods - from single parcels to containers full of items, perishables or non-perishables, and any other kinds of products. The RBAC system is what users must pass through before being given access to these records (see Figure 3.2).

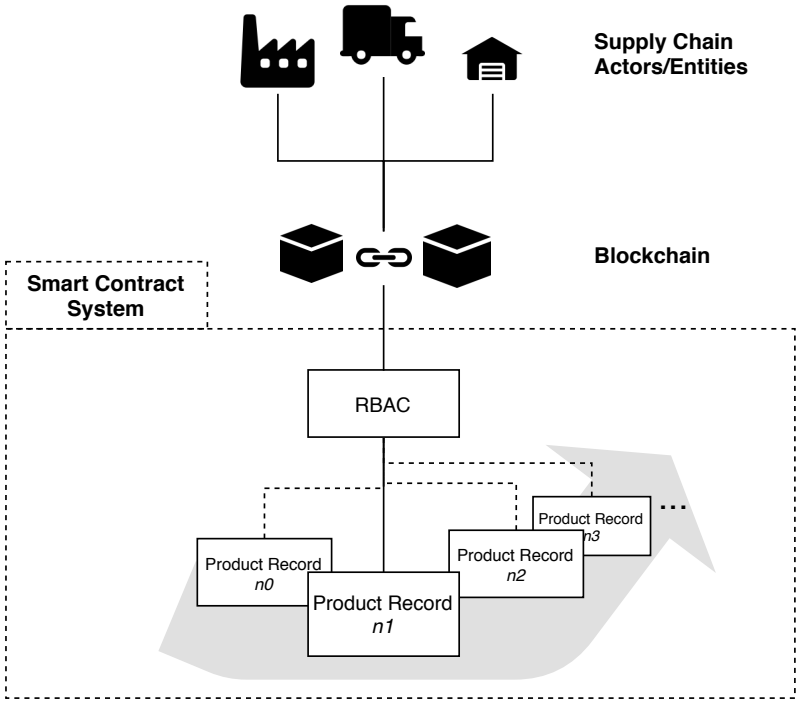


Figure 3.2: View of the general system structure. Users must pass through RBAC authentication to access product records.

It should be pointed out that in earlier stages of the system design, the Product Record was actually a form of digital asset (essentially a digital representation of the real product), and its legal ownership could be traded and transferred. This feature was eventually removed because it bloated the implementation, made the system more confusing and took the focus off of physical product tracking while putting it more on product ownership, which wasn't intended for the main themes this dissertation approaches.

3.3 The Product Record

When users want to track a product using this work's smart contract system they must first create its Product Record object, a digital record/registry for all of the data that refers to that product. Upon creation, the record is automatically given an ID number, which is the key used for accessing this structure. A field for an EOC timestamp denotes if the product has reached the end-of-chain and at what time.

Figure 3.3 shows how the Product Record was structured in terms of state data storage. Fields for tracking data, clearance/certification data and label data exist separately, but the first two particularly are not independent of each other. The tracking and certification/quality checks mechanism function in tandem throughout the system's implemented methods. Over the next sections these different fields of the product record and the processes associated with them will be discussed in more detail.

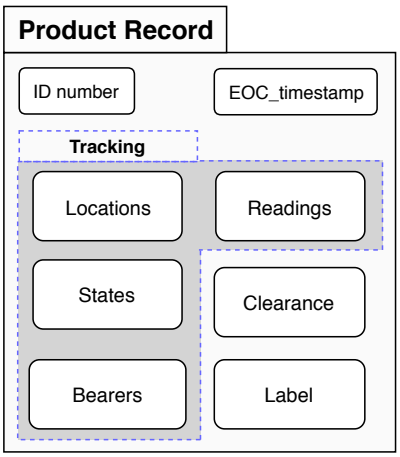


Figure 3.3: The product record structure and its state data fields.

3.3.1 Product Tracking and Tracing

One of the main purposes of this work is to process and maintain in a decentralized ledger the tracking and tracing data of items that are travelling through supply chains. The concepts of tracking and tracing are very similar to each other, their main distinction being on time relativity - tracking is often associated to real-time event-watching, while tracing is more related to reconstructing events from present-time to a point of origin. Both refer to the pinpointing of an object's location and other relevant data being in an ordered time sequence.

When deciding on what kind of events the system should be registering and what information must be collected, we came up with four basic questions that the system should be able to answer for every product registered on it:

1. Where was the product located, where did it pass through?
2. Who produced it and who held it during its journey?

3. What processing states has it gone through?

4. What was its condition during the journey?

And of course, the answers to these questions must have a timestamp associated to them, so that they can be reconstructed in a timeline in the form of events.

Taking this into consideration, it was decided that the system should be registering logs on four kinds of events: location changes, state changes, bearer changes and sensor reading data, all saved to separate structures (see Figure 3.3). They are detailed as follows:

- **Locations** - The product's location is arguably the most important kind of tracking data that can be collected, particularly to logistics entities. It is key to supply chain management that they know at all times where the items for which they are responsible are located.

  Location logs register the author and time of the log, the location's coordinates (latitude and longitude) and a descriptive name of the location. This data is enough to make approximate reconstructions of the physical path the product has taken on its supply chain journey.

  States - Coded changes in the transport/processing of the product's state are also registered on the blockchain, along with the log's author and a timestamp. The coded state contains information on what stage of supply chain processing is the item going through, or has gone through (i.e: loading authorized, crossed border, arrival at port, etc.). State tracking logs were implemented with the UNECE's Status Codes for Transport and Trade in mind [57].

- **Bearers** - The product's bearers are the entities that have physically held the product as it travels through the supply chain. Any change of hands is registered and timestamped. The current bearer of a product is a kind of sub-role in itself, because to make changes to a product record, a user must be its current bearer. This role is limited in scope to a specific product record, however, while the roles managed by the RBAC system have system-wide permissions.

- **Readings** - Any sensor data read from IoT devices is registered as a reading, which is a generic data structure that stores numeric values, and has a coded 'type', referring to the kind of reading it is (i.e average temperature, average humidity, weight etc.). They are meant to store any measurable values on the item's conditions, and like other tracking logs, have a timestamp and author associated to them. Government/customs authorities or certifiers can also register logs of this kind, without having to be the product's bearer.

From having these four sets of data being input into our system, a detailed timeline of the product's journey through the supply chain can be effectively reconstructed.

3.3.2 Clearance / Quality Control and Certifications

One of the features this work is meant to implement is semi-automatized quality control or clearance for products, as well as a registry of certifications. To this end, there exists a structure called Clearance in the product record (depicted in Figure 3.4) that stores certifications and terms/rules for quality control associated to that product's approval by regulators.

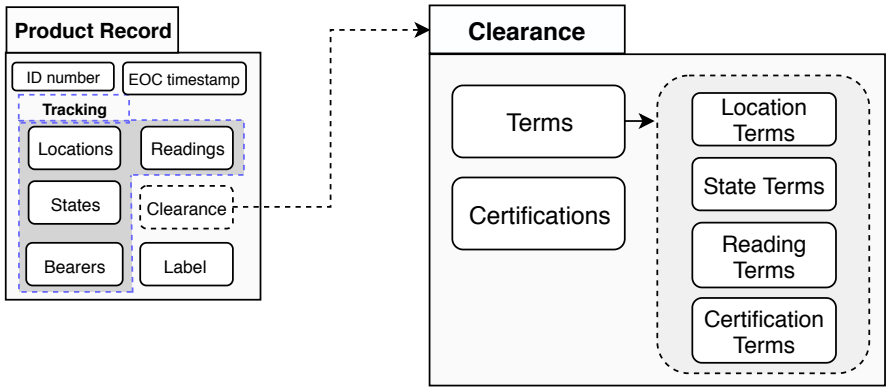


Figure 3.4: The clearance structure, which is part of the product record, and its respective data fields.

Terms for quality checks are essentially sets of rules that must be followed throughout the product's supply chain journey. Four different types of terms were defined: location, state, reading and certification. Table 3.1 lists the different terms implemented in the system's final solution.

Table 3.1: Different term types implemented in the system.

Term Type	average temperature	maximum temperature	minimum temperature	arrived at location	state included	state excluded	certification obtained
Code	10	11	12	20	30	31	40
Conditions for Success	Average temperature values are within a given range	Temperature is below a maximum threshold	Temperature is above a minimum threshold	Item arrived at a specific location	Item reached a certain state	Item did not reach a certain state	Item received a specific certification

Once terms are created, every time tracking data like a new location, state or reading is input, it is checked against terms of that type for that product. The same thing happens with the registry of certifications.

If a new location is registered, for example, the smart contract checks if there's any active terms for location on the product record. If indeed there is a rule (or more) saying that the item should be passing through a certain location, the latitude and longitude values being input by the user are evaluated against the ones set by the location term/terms. The term's state is changed to successful if the latitude and longitude values match, within a specified margin of error.

In this way, the product tracking system and the clearance system function in tandem, allowing for some automated quality checks to be verified by the smart contract system.

Users can also set time limits for terms being active. If a certifier wants the product to pass through a certain state during the next two days, he can create a term that is only active during that time, and which is automatically given as unsuccessful if that state is not reached within that time limit.

When the product reaches the end of the supply chain (also known as end-of-chain or EOC), the success of all the terms that were created is evaluated, with the item's overall clearance for the journey being judged successful or lacking according to that evaluation's result. This can enable logistics entities to automatically know if there are any problems in their supply chains.

The process described throughout this section, also aims to demonstrate that the capabilities of smart contracts for data processing and logic evaluations can be used in a system for logistics - particularly for automatizing some of the verification procedures that items in supply chains must go through, ensuring that they have been adequately processed and transported.

3.3.3 Labels

In the early stages of research for this dissertation, one of the ideas on the table was for the smart contract system to implement an already existent logistics tracking system, based on a single standard. Very quickly we realized that the wide array of different standards being used was potentially one of the reasons why the complexity of these systems increased.For example, one of the labelling that appears to be very used in the industry is the GS1 Logistics Label Guideline [58], but even within the GS1 standard there are numerous sub-types of labels that depend on the type of package being shipper. The spectrum becomes even wider if the different kinds of labelling for item types are considered (i.e. food, textiles, plastics, etc). It was concluded that attempting to implement a system that supported even only a few of these different standards would incur a level of complexity that was considered outside the scope of the work being developed here.

One of the ways found to partly support different standards was creating a field on the product record reserved for the label data. Instead of creating a large structure to store this data, which would be unintelligible due to the complexity of different standards it should support, it was decided instead that the label field would store a set of keys that point to the label data, which is being kept on a separate structure from the product record. If logistics entities want their data stored in a slightly different format than the ones that already exist, either because their product is different or the standards have changed, they can create these new structures in a new contract. A label number can similarly be used to obtain a product record ID number and through it the product record in question.

This approach can support a limitless variety of label types, of which there are numerous types, depending on the items being tracked.

3.4 Entities and Role-Based Access Control (RBAC)

To ensure that the tracking process would have adequate levels of trustworthiness and security, it was decided that our proposal should permissioned blockchain system - meaning only accessible to chosen parties. To this end, using smart contracts a Role-Based Access Control (RBAC) system was also implemented, in which a blockchain user's access and permissions are dictated by the type of role they have. Four main groups of entities involved in maintaining supply chains as the system's roles were identified. Their permissions to act in the system are laid out in Table 3.2, followed by a more detailed description of how these roles were characterized and the part they play in supply chain processes.

Table 3.2: Different roles for actors in the blockchain and their respective permissions

Permissions / Roles	Permission to become bearer	Change location[a]	Change status[a]	Create products[a]	Manage users[a]	Mark arrival at end-of-chain[a]	Create terms for clearance/ transport[a]	Certify a product[a]
Producer	x	x	x	x			x	
Transporter/ Warehouse	x	x	x					
Customs/ Govt. Authorities	x	x	x				x	x
Retailer	x	x	x			x		
Registrar					x			

[a]When bearer of product

Producers and Manufacturers They are the makers or growers of products. To properly trace products, their origin should always be registered as the beginning of the supply chain, which is why only these entities can create new product records. They also have permission to add quality requirements or terms of transport for their products.

Transporters and Warehouses Transporter companies are the carriers of products, and are the main contributors towards physical tracking of items. Warehouses are storage points for the items journey throughout the supply chain. Oftentimes companies provide both warehouse and transporter services, so there is no differentiation towards these companies in our system.

Certifiers or Government Authorities Entities that can provide certification or quality checks on items are given extra permissions as certifier entities. They can create rules or quality requirements for transport or storage of items, and register certifications they have given.

Retailers	Retailers generally represent the end-points of supply chains, and are consequently the ones that can deactivate updates to product records that have become obsolete, either due to sale or smaller-scale unit distribution. Besides being responsible for sale, they can also perform storage and transport duties, and therefore update tracking data as well.

Not contemplated in the above list is the registrar role, which exists for the sole purpose of managing user access to the system and their roles, and therefore doesn't have permissions to participate in the system's supply chain tracking.

The role of registrar exists as a consequence of building a permissioned smart contract system, intended for use in a consortium. Although the blockchain ceases to be fully decentralized by having its access dependent on a chosen group of authorities, it was judged to be a necessary measure to have users be pre-approved before being able to participate in the system. This pre-selection exists in order to create a level of trust and safety in authentication that is adequate to the requirements of logistics operations.

In consequence of this, to receive access to this blockchain application, user accounts must first be registered by a registrar in a list of trusted entities, along with their relevant data (name of company, contact and location address). As this is done, they will also be attributed a role that enables them to act in the system.

Almost all of the permissions to act in the system shown in Table 3.2 are also dependent of the user possessing the bearer role, which is specific to any one product record and therefore a kind of sub-role in the system. Being the current bearer essentially means one is in physical possession of the item and therefore can update its tracking status. Consequently, the roles defined in Table 3.2 come into action while the user is acting as a product's bearer.

3.5 IoT Device Integration

The integration of IoT devices in the system has already been hinted at in Section 3.3. Devices are one of the two parties (the other one is certifiers/government authorities) which can input Reading data onto the blockchain, basically logs of different types of measurements that are taken on an item's condition or the condition of its environment.

To interact with the blockchain, devices need to use a set of credentials, the same kind of private-public key pair that normal users utilize. Any entities authorized into the system can afterwards register in the blockchain system the account/credentials used by IoT devices under their ownership. Device applications use these credentials to send to the blockchain any readings acquired on products of which their owner entity is bearer. This means that while devices have their own user account to communicate with

the blockchain, they are subordinate to their respective owner entity and are not really included as actors in the blockchain tracking system.

Device owners can utlize the blockchain to send remote orders to their IoT devices in a master-slave relationship. The smart contract system therefore mainly serves as an authenticator for accessing data being collected by sensor devices. IoT integration with the blockchain system on the implementation level is discussed in Section 4.3.

3.6 Considerations on the system design process

There were two main challenges faced throughout the ellaboration of this work's smart contract system design:

1. The first issue was the complexity and variety of standards in logistics systems. Originally there was the idea that this dissertation's smart contract system was going to be based on existing logistics standards for transport and supply chains. However documentation on these standards proved to be quite complex, disorganized or even difficult to find. Ultimately it was decided that the smart contract system should support tracking functionalities that were generic yet practical enough to be implemented into almost any supply chain process.

2. Problems and limitations found during the implementation stages of this dissertation often led us to change some of the aspects of the system model, often leading us to add, remove or tweak functionalities. The final model described in this section is probably the most compact, simple version of these many design-to-implementation stages that we went through. The key objectives are still maintaned in the final solution, which successfully implements a form of decentralized tracking that takes advantage of smart contract technology's validation and logic capabilities to improve supply chain processes.

In spite of these problems, we judge that the overall result of the system design fulfills the objectives set out first on Section 1.2.3 and also afterwards in more technical detail, on Section 3.1.1.

It should also be noted that the system model presented here does not take into consideration privacy issues. Blockchain decentralized systems can only have a level of privacy when the users accessing the system are not known, which is not the case here. Taking into consideration that all data input on this system is supposed to invoke transparency and trust in supply chains, and that the data being inserted is not particularly sensitive (i.e. financial records or other documents that mustn't be exposed to the public), pertaining only to tracking changes that are, on some level, available in existing systems, we considered it was a good measure to keep the system as transparent as the technology allows. That being said, some precautions were taken to not make product ID numbers (which

give access to records) overly easy to access (see Section 4.2.1.1 for more information on accessibility).

The system model described throughout this chapter then serves as basis for the implementation of our smart contract system, which is explained in Chapter 4.

Chapter 4

System Implementation

This chapter details the implementation process of the system described throughout Chapter 3. First, a quick overview of the development tools and languages used will be given. Afterwards, the general architecture of the smart contract system and subsequent APIs developed will be given, and the individual components explained in more details throughout the rest of the chapter. The closing chapter will discuss the challenges found throughout the implementation process.

4.1 Ethereum Framework and Tools

The Ethereum blockchain framework was ultimately chosen for the implementation of the blockchain-based logistics tracking system proposed on this thesis. This decision came down to a few reasons:

- A wide range of development tools being available, as well as extensive documentation and an active community, point to Ethereum as being at the forefront of blockchain technology.
- Support for smart contract technologies with the highest level of complexity currently available , which allow for versatility in the processes being implemented.
- Support for building applications that function on top of blockchain technology, allowing regular users to to access data or interact with smart contracts via more user-friendly interfaces.
- A good level of portability of Ethereum smart contract projects, as there are many options in existence and development that utilize the EVM as their smart contract execution environment[27], or allow portability (ex. Hyperledger Sawtooth).

Ethereum uses the EVM to compile Solidity smart contracts into Ethereum blockchains and communicate with them. Solidity is a coding language was created for the sole purpose of developing smart contracts for Ethereum. As such, while the language is quite simple in terms of syntax, it contains a number of particular features (see Section 2.2.3), which mostly derive from the fact of smart contracts being kept and validated in a decentralized, account-based system.

For the development and implementation of our smart contract system, the main Ethereum development tools and libraries used were the following:

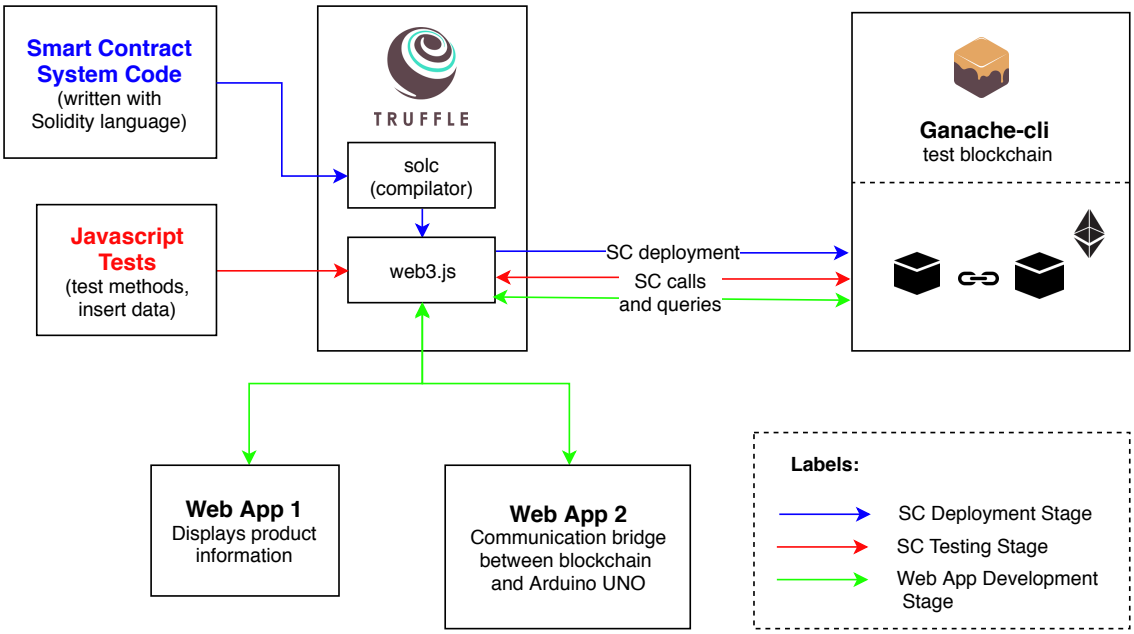


Figure 4.1: Framework for creating, compiling, deploying and testing smart contracts using Truffle and Ganache-cli.

- **Truffle** - an Ethereum smart contract development framework [59]. It permits easy compiling, deployment and testing of Solidity smart contracts in a chosen network, with some debugging features for transactions as well. Compiling is done with the solc compiler and deployment testing largely use the web3.js library, which is an important tool used also in the application development stage of this work.

- **Ganache-cli** - a lightweight, Javascript emulator of Ethereum blockchain networks, run as a local node [60]. It offers a wide array of options for test network customization, enabling developers to test their smart contracts in a private test environment that is made to measure before they are deployed in public or private blockchains.

  web3.js - an Ethereum Javascript API, web3.js implements the JSON-RPC protocol to connect and interact with any Ethereum blockchain networks [61]. It is a library dependency of Truffle as well.

These three tools and an IDE (Integrated Development Environment) are enough to develop Solidity smart contracts, deploy them on a customized blockchain and run some Javascript tests on them. Those tests can be used to validate the correct execution of smart contracts and measure the overall performance of the system in terms of transaction cost, time and data storage.

Additionally, some basic smart contract functionalities were implemented using open-source contracts made available through the OpenZeppelin[34] library, which provides standard smart contract code to be used in all kinds of smart contract systems. This work specifically uses the RBAC.sol smart contract from OpenZeppelin v.1.12 , which implements the basis for our Role Based Access-Control system.

The tools used for the second part of this work - the development of the two browser applications referenced in Chapter 1 - are described in Section 4.3.

4.2 Smart Contracts System Structure

For the explanation of the smart contract system implementation in this chapter, we start out once again by discussing the general structure of the final system that was implemented and then explain in more detail each one of its features.

One of the functionalities Ethereum smart contracts implement is inheritance. When inheritance is used, it allows descendent smart contracts to access methods and data structures of their parent contracts. This means solidity code can be structured in a kind of hierarchy, which is shown in Figure 4.2. The final contract launched in the blockchain in our case is the 'Product Manager', but it inherits all the methods and structures of its preceding ancestor contracts as well.

The development of our smart contract system was done above all using inheritance, which served also to separate different smart contracts largely according to each one's functionality. Halfway through the development and implementation process, some issues started occurring with the smart contract system's deployment. It was eventually realised that the problem lay in the size of the code being deployed, and in the way that the system was conceived initially.

The drawback of inheritance is that the final code of the contract being deployed includes both the code of the ancestor and child contracts. This led to the child contract Product Manager becoming too big, exceeding the storage size currently allowed on any one Ethereum block. A different implementation could solve this issue by having these smart contracts be deployed separately into the blockchain and communicate via interfaces. This code size problem was found at a stage where altering the system to work solely through interfaces seemed to pose a challenge in of itself, particularly because most of the methods we wanted to implement, which revolved around the product record structure and functionalities described in Section 3.3, converged into a single contract, Product Manager, and were tricky to separate due to co-dependence.

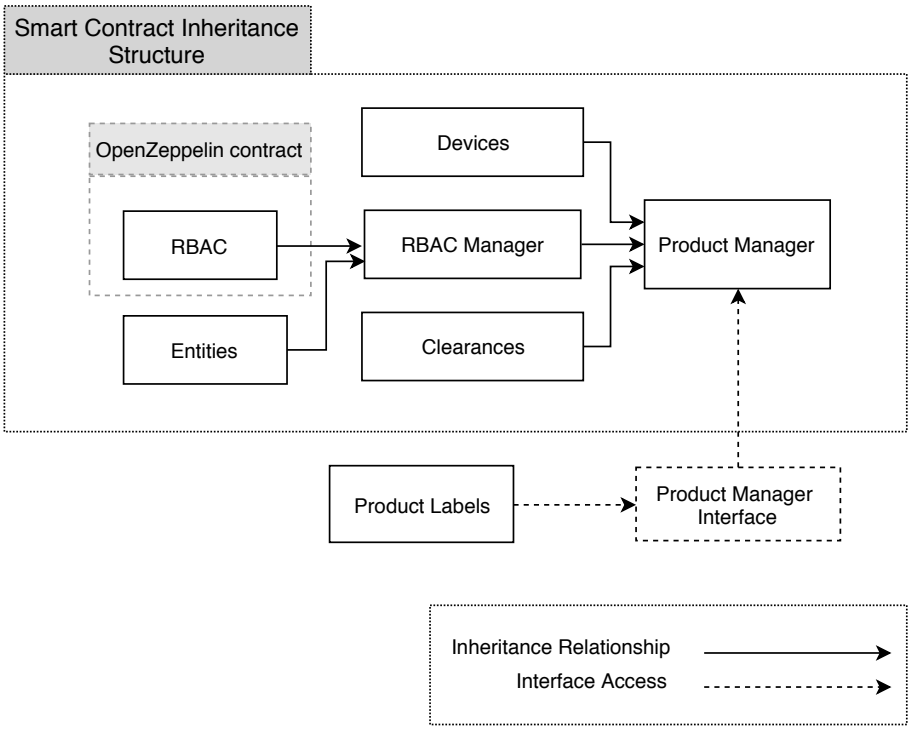


Figure 4.2: Smart contracts implemented through inheritance system and interface access.

Therefore it was decided that priority would be given to fleshing out the smart contract system, implementing all the functionalities we wanted to, building and integrating the applications we planned to make for the system and only eventually deconstruct and separate the code with interfaces if there was time. Although this task had to be left for future work, interface functionalities were implemented in the case of the system for support of different label types, which was easily completely separated from the other components of the smart contract as it was designed to be a simple and adaptable feature.

The smart contract system is separated into the different modules seen in Figure 4.2 in a way that separates the four main systems we wanted to implement: RBAC authentication for entities, support for different labels, a quality control and clearance evaluation system and a product tracking system. The first two are systems are fairly independent of all the others which is why RBAC-related functionalities can be put at the top of the inheritance tree and the labels system can be easily implemented without inheritance at all. Product tracking and quality control and clearance are much harder to separate because their mechanisms are interwoven and also very dependant of RBAC. Each of these smart contract modules and their purpose can be quickly summarized to give a better wide-view of the system:

- **RBAC and RBAC Manager** - RBAC is a smart contract taken from the OpenZeppelin library that implements basic methods and structures for attributing and authenticating any kind of string-defined role. RBAC Manager limits the creation of these roles to the ones we defined in Section 3.4, restricts role attribution to only

users that have already been registerd as entities, and makes it so only the registrars of the system can manage entities and role removal and attribution.

- **Entities** - A smart contract for registering authorized users of the system and storing important data relative to themselves or their business - namely a descriptive name, a contact and an address.
- **Device** - A smart contract for registering user device accounts into the system.
- **Clearance** - The smart contract that contains the Clearance, Certification and Term structures, as well as some of the methods for term and quality control checks.
- **Product Manager** - The central smart contract to the system, where the product record structures and state exist, as well as the product tracking system and clearance checks, which utilize the RBAC system for authentication. Being the convergence point for all these systems, the Product Manager also contains the most methods and largest amount of code.
- **Product Labels and interface** - Contract or contracts containing the different data structures for storing diverse types of standards of label data, which are associated to product records.

The following sections describe in more detail the mechanisms and methods used for each smart contract's state and code.

4.2.1 Data Storage

Most smart contracts have a number of state variables defined in their code, and which are kept in their storage. They tend to serve at least one of two purposes:

1. To store the smart contract's state data, which controls the way it responds to calls and transactions - the same way a state machine's variables control its behaviour;
2. To store important data in a decentralized ledger, guaranteeing its untamperability.

The use of state variables comes at a high cost of computing and storage resources, however, which is why writing and rewriting smart contract storage is the most expensive kind of transaction that users can make in an Ethereum blockchain [8]. Consequently, one of the main concerns of designing a data registry-oriented smart contract system such as ours is the optimization of data structures and methods with which data is organized and stored. This must be done in a way that the cost-efficiency relation of using the system can be optimized, while also taking into account data accessibility.

To this end, it is important to know the two main types of large data structures that can be used to store values in the Ethereum blockchain: arrays and mappings.

Arrays are iterable, but whenever one wants to make a deletion and keep the same order of elements, a varying number of array members have to be shifted or rewritten into the right position. This can make simple operations unpredictably costly.

On the other hand mappings, which are essentially non-iterable key-value hash maps, are slightly more expensive to access and write on than normal arrays, but members can be added and deleted independently of each other and at a constant, well known cost for each operation. Additionally, if developers want to iterate a mapping structure, they can always store their key values in an array. Using both array and mapping structures can therefore be used to guarantee data accessibility isn't lost, at the cost of more storage usage. For these reasons, mappings were used to store and make accessible most of the data structures on our smart contract system.

4.2.1.1 State Variables and Accessibility

All of the smart contracts mentioned in the previous Section 4.2 have state storage variables in use. The exception is the Clearance contract, which contains only class structures and some methods. Table 4.1 lists and summarily explains the purpose of the state variables present in each smart contract. State variables with an arrow (→) associated denote the use of Solidity mappings, while brackets ([]) refer to arrays. Variables with names starting in upper case letters refer to class structures (the Product Record for example was described in some detail in Section 3.3).

Table 4.1: State variables implemented in each smart contract.

Smart Contract	State Variables	Description
RBAC	`role_name → (user_address → has_role)`	A double mapping state variable. The first key is the role_name, which leads to a mapping of user addresses to a boolean value. The boolean value reveals if the user has the role of that name.
RBAC Manager	`role_names[6]`	List of role names allowed in the system. It is a constant variable.
Entities	1. `entity_addresses[]` 2.`user_address → Entity`	1. List of user accounts/addresses registered into the system 2. Mapping of user accounts to their respective Entity data structure, which contains a name, a contact and an address.

Devices	1. device_addresses[] 2. device_address → Device	1. List of device accounts registered into the system 2. Mapping of existent device accounts to their device data, which includes their owner and name/description.
Label	1. Label_structure[] 2. label_number → product_id	1. Array of label data structures registered in the system 2. Mapping of the label number to product id values (used to access Product Records).
Product Manager	1. product_id → Product Record 2. product_count 3. user_address → bearer_products_list[] 4. user_address → producer_products_list[]	1. Mapping of product ids to their respective product record. 2. Number of products in the system. Used to create product ids. 3. List of ids of items of which user is bearer. 4. List of ids of items of which user is producer.

The state variables of the RBAC and RBAC Manager smart contracts serve to register user's roles into the system. They are accessed every time role authentication or management is needed.

As for the Entities and Devices smart contracts, their state variables are essentially data registry structures - one for regular users information and another for devices, although they can be used for authentication purposes as well. The address keys to access these data structures are kept in iterable arrays.

The user_address variable in particular is commonly used as a key value for accessing structures. This is because each user can be identified through his address variable, which is available to the smart contract on every transaction/function call that the user attempts to make. So while there is a list of user addresses registered into the system in the entity_addresses variable, it is generally only used for registry authentication.

The other reason this address list exists is so that the key to certain values is never lost - particularly, so that product id numbers and their respective product records can always be retrieved. Figure 4.3 illustrates the main key-value relations between our smart contract system's state variables. In case a registrar wants to search all of the product records on the system, for example, they can use the addresses kept in the entity_addresses variable to find all the IDs and associated product records structures that are in the system, via etiher the producer or bearer product lists. This can be important in the eventuality that smart contract state variable storage must be deleted/freed. The keys for access of storage structures should never be lost.

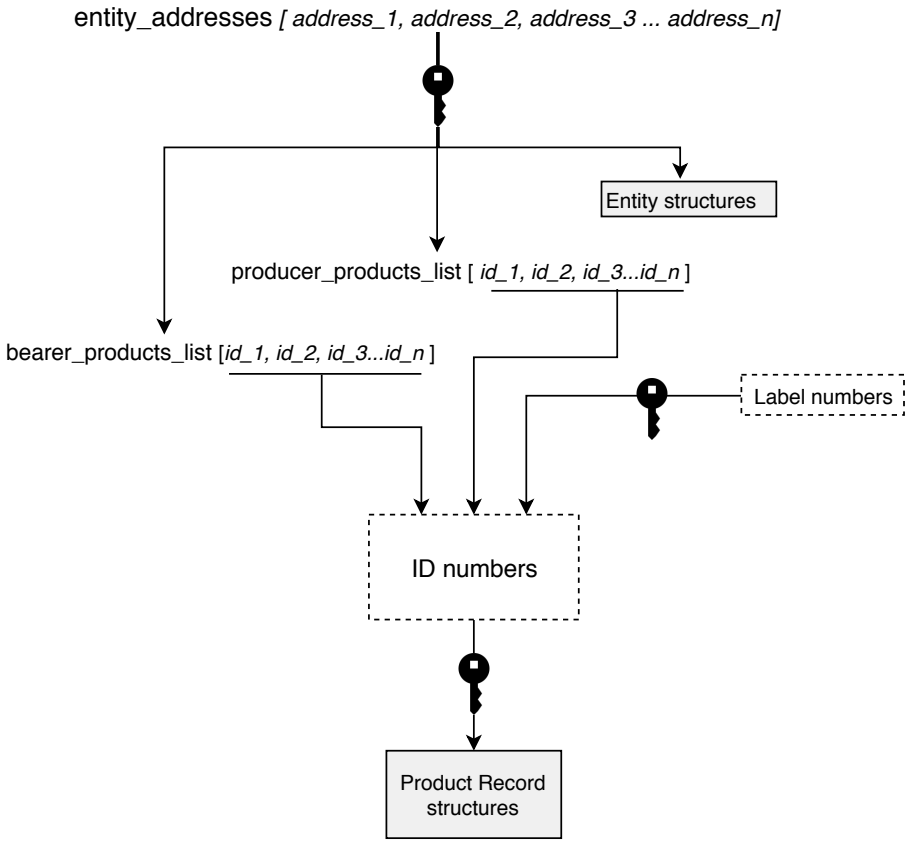


Figure 4.3: Diagram of key-value relationships that give access to the Entity and Product Record data storage structures

The Product Manager smart contract implements all of the state variable and data structures associated to the product record. A product's tracking data is accessible for viewing through the product's system ID, the key value to access a product record structure.

As for product ID numbers, they can be obtained through the label number or, for producer and bearer entities , one of two mappings of addresses to ID lists. These lists contain the ID of every product each producer has manufactured or of every product currently in possession of a bearer.

These lists exist so that whenever entities use our blockchain application, the smart contract uses their address to immediately obtain the data on the products which they have produced or are in possession of. In this way, producer and bearer entities do not have to insert product IDs themselves to access tracking data - instead, the application identifies them and retrieves this list automatically.

4.2.2 Methods and Functionalities

When designing this smart contract system, there were three main functionalities we wished to implement: RBAC, product tracking and quality control methods. The state

variables discussed in the previous Section lay out the groundwork for achieving these objectives, which can now be fulfilled with the implementation of methods that dictate user access to the smart contract system.

To help visualise how the implementation of these features came together, Listing 4.1 shows an excerpt of the Solidity code used to implement some of these features on the Product Manager contract. It contains the Product Record structure we defined in Section 3.3 of Chapter 3, as well as a function for updating product tracking data (insertion of a reading). Some parts of the code are omitted for easier reading.

```
1  struct Product{
2    uint32 id;
3    uint32 eoc_timestamp ;//end of chain reached timestamp
4    Label label;
5    State[] state_log; //coded state changes + a timestamp
6    Location[] path;//(lat,long) points, name and a timestamp
7    Bearer[] bearers;//bearer address + timestamp
8    Reading[] readings; //stores readings of tracking info used for clearance checks (temperature,
         weight etc)
9    Clearance clearance;
10 }
11 mapping (uint32 => Product) products; //id to product record mapping
12
13 function addReading(uint32 _id, uint8 _term_type, int32 _final_value, uint32[2] _timestamps) external
       productExists(_id) productSupplyActive(_id){
14
15   require(hasRole(msg.sender, "certifier") || addressToDevice[msg.sender].owner == products[_id]....
         bearer_address, "Only the bearers devices or customs entities can register readings"); //exit
         function if it isn't being called by an authority or one of the bearer's devices
16
17   Clearance storage c = products[_id].clearance;
18
19   for(uint i = 0; i < c.misc_terms.length; i++){
20     if(c.misc_terms[i].term_type == _term_type){ //check for terms/rules on this type of reading
21       if(...|| c.misc_terms[i].period_timestamps[1] > _timestamps[1] ) {//checks time constraints
22
23       c.misc_terms[i].success = _evaluateMiscTermSuccess(c.misc_terms[i], _final_value);
24       break;
25       }
26     }
27   }
28   products[_id].readings.push(Reading(msg.sender,_final_value,_timestamps,_term_type));
29 }
```

Listing 4.1: Excerpt of some code on the Product Manager contract. It defines the Product (product record) storage structure and the addReading function.

The `addReading()` function is a great example of all of the smart contract system's functionalities being used to evaluate, validate and execute a task of storing tracking data. The function has two modifiers, `productExists()` and `productSupplyActive()` to check if the user call being made is on an existing, active product record. Right afterwards, in the contract code, with a `require()` statemenet, the smart contract checks if the the user has permission to add reading type data to a product record - which per our system model (see Section 3.4) has to be either a certifier/authority entity or a device registered under the current bearer.

If all these checks have been passed successfully, the smart contract then accesses the Clearance structure of the product record to check if there are any terms of the "misc"type

(meaning they are intended for readings, which can have diverse types). If a term is found, and its type matches the reading's, and the function call is being done within that term's time constraints, then another function is called to evaluate the reading's input values against the term requirement. The success of this evaluation may or may not change the success state of that term. After this process is finished, the new Reading is added to that product's readings array log.

All of the methods for interaction with the blockchain network were implemented following the same kind of process seen above, with RBAC processes authenticating users and tracking data evaluations being done upon input.

Besides methods that interact with the blockchain network, we also had to implement and array of view-type functions, which are needed for applications and users to access and visualize the contract's storage. Since they don't alter the contract's state, these methods are completely free to call and have no cost to the users.

4.2.3 User Interaction

After getting a low-level look at the smart contract system's functioning in the two previous sections, we can look at the system's process through a higher-level outlook - like a user would.

Figure 4.4 shows how the tracking of a product unfolds as it travels through the supply chain in three different views:

- On the supply chain level, we see what methods each bearer entity uses to introduce data into the blockchain. Changes in state, location or bearer are marked by a diamond, circle and square shape, respectively.
- On the tracking level, we see how the data introduced by the bearers can be used to reconstruct a timeline of events for changes in state, location and bearer, as well as any readings received from IoT devices.
- On the level of clearance checks, we see the terms of transport laid out by the producer or government authorities. We also see at what points the data introduced into the blockchain was checked against these terms, and whether it fit the requirements they asked.

In this example, the producer starts out by introducing the product data and its location information into the system with `createProduct`. In this way he becomes the first bearer and the point of origin for this item. He updates the product's state accessing the `addState` function, and creates a term for clearance, T1, related to its state - in this case, a requirement for the product to pass through a certain state before it arrives to the end-of-chain. The producer then uses the `changeBearer` method to pass the product on to Transporter 1.

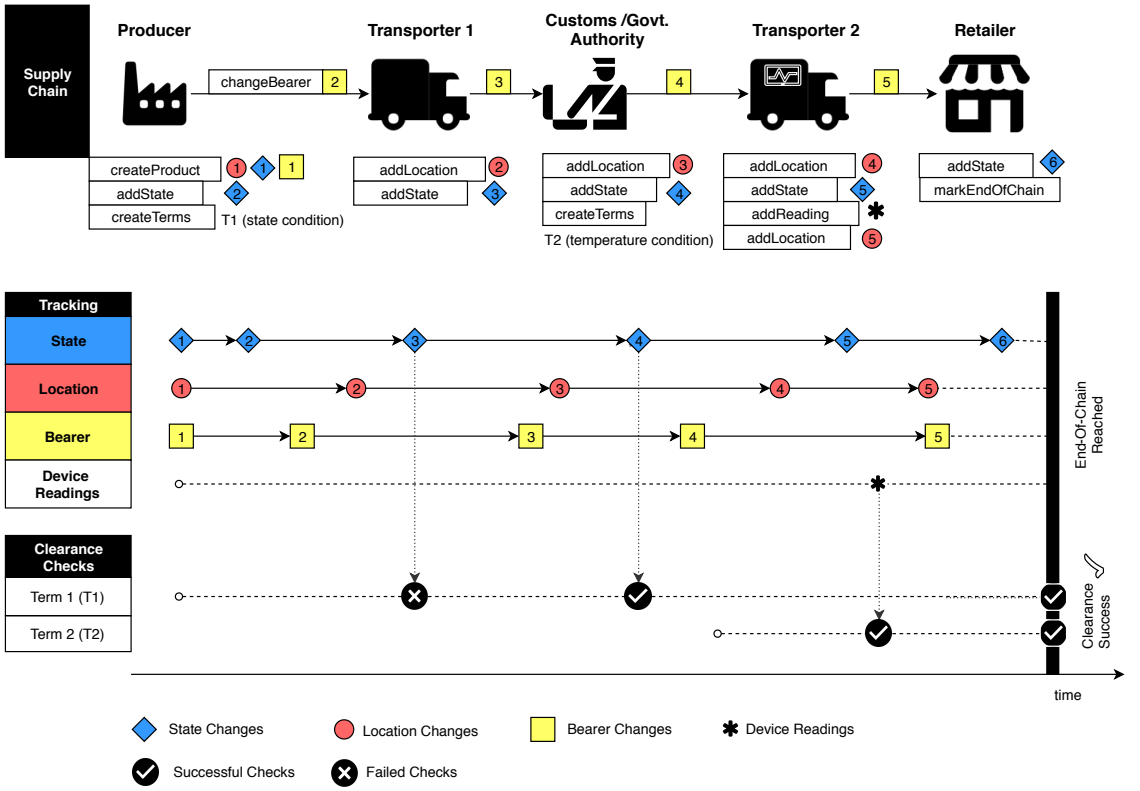


Figure 4.4: Representation of the tracking system and the methods implemented, showing an example of a product's journey through the supply chain and a timeline of events.

The products tracking data continues being updated, and state changes checked to see if they fit the requirement of T1. A government authority eventually becomes the products bearer and makes a state change that fits the requirement for term T1. He also creates a new term for transport, this time a temperature requisite. As the product changes hands to Transporter 2, an IoT device (temperature sensor) associated to the bearer transporter entity monitors the temperature conditions of transport and at the end of the journey sends the value of its readings to the blockchain. These are checked against term T2 and turn out to fulfill its requirements.

As the product reaches the end-of-chain, it has passed all its clearance checks. A record of its journey exists in the blockchain in the form of event logs for state, location, and bearer changes, as well as device readings, certifications (not represented here) and terms of transport fulfilled. This information can be used to reconstruct the timelines seen on the lower half of Figure 4.4. It can be read from the blockchain using view type functions, with not cost to the users who call them. All of these user interactions happens through a browser application we created, and which we discuss further in the following Section 4.3.

4.3 Web Application Development

To utilize the work that has been developed up to this point in real-world case scenarios - such as the ones described in Sections 3.1 and 4.2.3 - normal users would have need of an API that would allow them to interact with our smart contract system.

The developers of Ethereum technologies have invested particularly in tools for building decentralized web applications (which they call Dapps [23]). These applications access specific smart contracts deployed to the Ethereum blockchain, and users spend ETH currency to interact with them via transactions/function calls. The same tools used for building Dapps can be utilized for applications that are connecting to private Ethereum blockchains, such as the one our smart contract system is projected to run on.

Therefore, for the work contemplated in this dissertation, a simple web application for accessing and viewing data stored in our blockchain smart contract system was developed. The deployment of our smart contracts to a test blockchain was done with the same tools detailed in Section 4.1 and Figure 4.1. For the web applications specifically the following technologies were used:

- **web3.js** - the same Ethereum Javascript API that is used to test our smart contract system can then be used for web development. web3.js is used to access the smart contracts in the blockchain and call their functions. For the development of smart contracts, the provider for the blockchain connection and the web3
- **Metamask** - to communicate with blockchain applications, users need to provide their user account credentials to the application. To ensure this is done safely, it is typical to use a kind of software called wallets, which store these credentials and give access to them in a secure manner. Metamask is one such software that exists in the form of a browser Add-on. It allows users to connect to various blockchain networks and also to log into and switch accounts easily, which is a useful feature for testing applications.
- **http-server** - a simple package for creating local HTTP web servers for application development.
- **Bootstrap.js and Morris.js** - the front-end development used Bootstrap.js templates/components and Morris.js for creating timeline graphics of product tracking events.

Testing and development of the application was done on the Chrome browser. The final result is shown in Section 5.2.

4.4 IoT Device Implementation

There are mainly two kinds of IoT devices that can be integrated into the system: sensors, meant for collecting data on measurable conditions like temperature, humidity, weight and the like, and actuators, which upon receiving certain signals trigger the activation of processes to be put into motion. While there were plans to implement actuators into the system, the work done here ended up only addressing the use of sensors for readings. However, theoretically the same kind of system used for communication between sensors and blockchain could be used for actuators as well.

When considering the implementation of IoT devices into our project, first, we must address the issue of where to run our blockchain client. Most IoT devices, due to energy consumption and cost constraints, are designed to have low processing power. However blockchain clients are heavy applications that require large amounts of storage and processing power to function. Consequently, currently the most straightforward way to connect an IoT device to a blockchain is through a gateway unit that runs the blockchain client and communicates with the IoT device via another protocol.

Secondly, as explained in Section 3.5, each device has its own user account or private and public key pair to interact with the blockchain. We must then consider where we store the private key, which is used to sign transactions and spend the account's currency. Knowing that handling transactions is demanding in terms of processing power, and that storing a private key in the device itself might prove unsafe when there's no access control on its memory storage, we chose to use a blockchain wallet[62] software in the gateway unit itself, which can enforce better security for this key.

Lastly, our application needs to know when it is supposed to send readings to the blockchain, of what type and on what products. To this end we implement the event log feature of Ethereum smart contract technology, so that the owner of a product can remotely send orders to start/stop reading values and to add them to the blockchain. This event system was designed to give orders to sensor devices through the blockchain - the orders are simply 'start reading', or 'stop reading', and enforce a master-slave relationship between the owners of the device and the device itself. This system makes use of our smart contract system's validation authentication functionalities, which assure that the orders are being given by the device owners, and refer also to items in their possession. The process we've just described unfolds in the following way:

1. A government authority creates a term for storage of an item. When a warehouse receives it, becoming its bearer, it accesses a method in the Product smart contract that fires a `StartReadingOrder` event for one of its devices, (if all bearer and ownership permissions are met).

2. The gateway unit is listening to the blockchain for events sent to devices that are connected to it. When the order to start reading is received, our application starts

storing the device reading information that was requested, and fires its own `OrderReceived` event, so the bearer can know its request is being attended to.

3. Before the product is shipped to another location, the bearer fires a `StopReadingOrder` event. Upon receiving it, the gateway unit finishes its stops its reading process and registers the data collected on the blockchain, under that product's tracking data. It is automatically checked by the smart contract against the terms for clearance created.

Figure 4.5 illustrates the connection made between the device, a gateway unit, blockchain and the device's bearer, including the software we used to bridge the communication gap.

We decided to use an Arduino UNO microcontroller and an Adafruit SHT31 temperature and humidity sensor as the data collector. This temperature sensor communicates its values to the board via an I2C connection, and its setup with Arduino is very simple and can be found on Adafruit's documentation for the sensor [63].

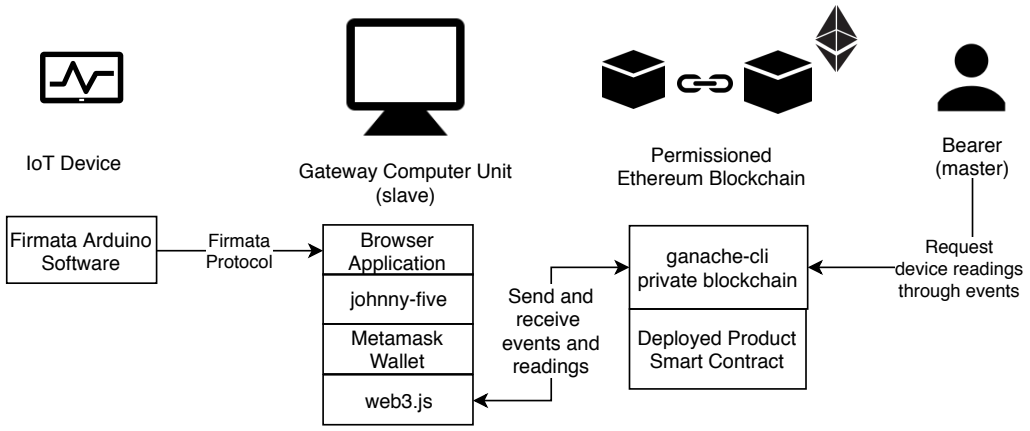


Figure 4.5: Diagram illustrating how we collect sensor data from an Arduino and send it to the blockchain.

The Firmata protocol [64] is used for microcontroller - computer software communication. Although the tests we made utilized a USB connection, the protocol can be employed to wireless connections as well, provided the microcontroller in question can connect to Wi-Fi networks.

The bridge between device and blockchain is made through a browser application in the gateway device. In this application, the johnny-five library interprets the pin output of the Arduino board that is reveived via Firmata connection. The browser application was made with Node.js technologies, and uses the socket.io[65] and dweet.io[66] libraries to create a socket connection for Firmata and then emit the data received real-time.

The web3.js[62] library is used to connect to private Ethereum blockchain, where our smart contract is deployed, and uses the JSON-RPC protocol to communicate. Metamask is used to manage the device's user account and transaction signing.

4.5 Considerations on the implementation process

Throughout the implementation process of the smart contract system a number of difficulties and setbacks were found. They can be enumerated as follows:

1. The most notable problem found has already been mentioned in Section 4.2, and relates to the code size and complexity of the total smart contract system. Halfway through implementation, our code started to hit the size limit that smart contracts are currently constrained to when being launched in the Ethereum blockchain network. The solution to this issue lays in separating the smart contract code into chunks and use interfaces instead of inheritance to access each contract's methods and structures. However for the code complexity of our system, adopting this approach would have required considerable rewriting/ rearrangement of the work done until then. Considering the time frame and goals that had been set out for this dissertation, we decided to first successfully implement all the functionalities that had been envisioned for this smart contract system and eventually, if there was time, refurbish the code into separate contracts that used interfaces and didn't surpass the code size limit. This feature has in this way been left for future work considerations.

2. Solidity and the EVM, while able to support complex logic in coded systems, have nonetheless some limitations in terms of features, particularly when compared to other programming languages. One that comes to mind is the lack of support for either fixed or floating point numbers, which led us to store values such as latitude and longitude as integers, and having the user API handle the conversion of integer values to decimal numbers. This feature might eventually be supported in future versions of Solidity. Some other functionalities like limitations in string operations and in the return values allowed for functions had to be worked around.

3. While there are limitations to Ethereum technologies, the fact is they are being developed at a rapid pace. The solution developed here used Solidity v.0.4.24, but throughout these months a more recent stable version came out, v.0.5.5, and it is already in use. The web3.js and OpenZeppelin libraries went through similar updates. This sometimes brought up some incompatibility issues, so the solution implemented here used above all the older versions of these tools. This means however that in the short six months this implementation was developed, the technologies used here are already being replaced with improved solutions.

In spite of these troubles, the final implementation has fulfilled the objectives that were demanded of it. Since it is being developed with application to a private blockchain network in mind, the network constraints that were found, although important, are not unmanageable or unsolvable - Ethereum technology's fast development and growth is a good indicator on this. What is left to know is how the system would fare in terms of

scalability and cost, which is something that will be explored in the following Chapter, discussing results.

Chapter 5

Results

In this chapter, an evaluation of the system developed and implemented will be made. The tests shown here will first determine if the smart contract system is behaving as it should upon interaction by users with its methods. Then, an analysis of the system's costs and scalability will be made in order to determine if the blockchain-based solution presented here could be used on real-life applications. Afterwards, an overview is given of the web applications developed, the first for accessing blockchain data and the second to have a temperature sensor send readings to the blockchain system.

5.1 Smart Contract System Testing

We tested out the most important methods and functionalities of our smart contracts using Truffle and a local blockchain, generated with ganache-cli, (see Figure 4.1). For calculating and displaying the costs of deploying our smart contract system and calling its methods, we used eth-gas-reporter[67].

5.1.1 Testing Functionality

The first tests that need to be done to our smart contract system are on the success of its implementation. The functionalities we have described throughout the last chapters should be effectively achieved, namely:

- The RBAC system must authenticate users and bar access to functions if they don't have the appropriate roles;
- Storage and access of tracking and certification data must be done correctly;
- Quality checks and clearance mechanisms evaluate tracking data as expected.

So to determine if these processes were indeed working as they should, we set about testing the smart contract's methods.

For testing the RBAC system what these tests did essentially was have a registrar user attribute different roles to a number of other user accounts. These user accounts were then used to test the several different methods that altered the contract's state. It was verified that indeed if the user calling certain methods did not have the appropriate role or permission to do so, the blockchain network would reject his transaction.

Afterwards, view type functions (which access state variables, but do not alter them or execute transactions) were used to retrieve values from the blockchain system, and verify if the smart contract's state was being changed as expected, overall simulating a scenario such as the one seen in Fig.4.4.

Listing 5.1 shows part of the code used on one of these tests. Some of the function input is omitted (marked with (...)) to make for easier reading. The smart contract's state is changed, then read, and we compare the blockchain's output with the expected output values.

```
1  //registering state, bearer and location changes in the blockchain,
2  await contractInstance.addBearer(...{from: account1})
3  await contractInstance.addProductState(...{from:account2})
4  await contractInstance.addLocationToPath(...{from:account2});
5
6  //registering a device reading and marking end of chain
7  await contractInstance.registerReading(...{from:account3});
8  await contractInstance.markEndOfChain(...)
9
10 //collecting the state data back from the the blockchain
11 var clearanceResults=await contractInstance.viewClearanceSuccess(ids[2]);
12 var productInfoB = await contractInstance.viewBearers(ids[3])
13 var productInfoL = await contractInstance.viewLocations(ids[2])
14 var productInfoS = await contractInstance.viewStates(ids[2])
15
16 //checking if our input was registered and if clearance was processed correctly by comparing the
       blockchains output with our expected output
17 assert.equal( clearanceResults.toString(),expected1.toString())
18 assert.equal(productInfoB.toString(), expected2.toString())
19 assert.equal(productInfoL.toString(), expected3.toString())
20 assert.equal(productInfoS.toString(), expected4.toString())
```

Listing 5.1: Excerpt of Javascript code used for testing the methods related to tracking and clearance.

Tests on this front were successful and showed that data was being stored as expected, with state/tracking changes and clearance or quality control being processed properly. RBAC features also worked correctly.

5.1.2 Gas Costs

In Ethereum blockchains, the cost of smart contract deployment and usage is measured according to the type and number of computer operations that are done when using them. This has been explained Section 2.2.4, but to summarize, we consider two variables when dealing with smart contract cost:

- **gas cost** - the cost in gas units of executing computing operations in blockchain smart contracts;
- **gas price** - the variable that defines the price users must pay per gas unit, in ETH currency.

Gas cost is a deterministic measure, meaning that when executing a method several times, with constant initial state conditions and using the same input, the execution process's final gas cost is always the same.

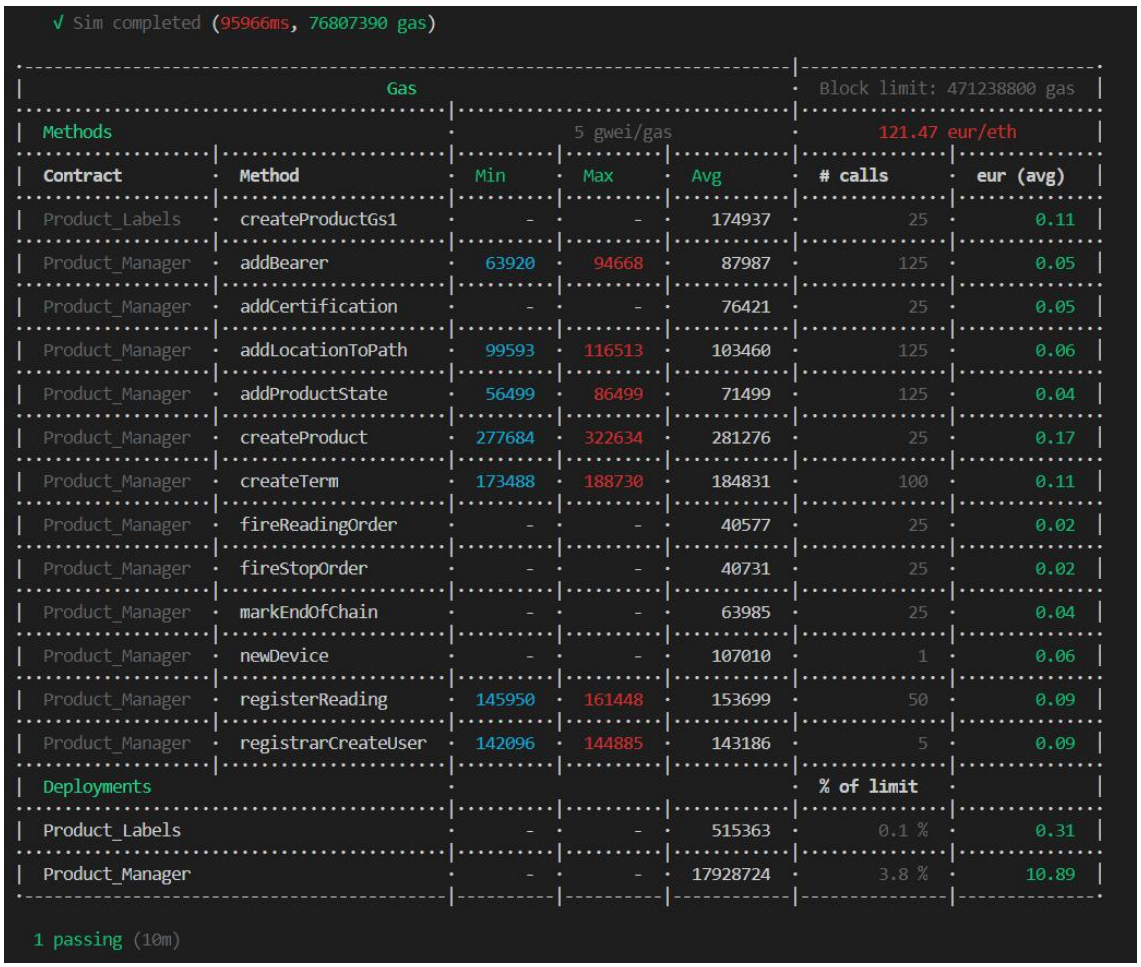
Gas price on the other hand is set by the user calling the method, and represents the value he is willing to pay to the blockchain miners, per gas unit, for including his transaction in a new block. Multiplying a method's gas cost for the gas price offered by its caller gives us the amount of ETH the user will pay the miner. ETH cryptocurrency prices can then be converted into normal currencies to give a better notion of real cost behind that operation. These prices are only relative to usage of the public Ethereum blockchain. It should also be noted that the ETH market fluctuates heavily, so in the coming months both the gas and ETH prices used here can become outdated.

To obtain average gas cost values for executing operations in our smart contract system, a test was developed for creating a set number of products and simulating tracking data insertion for each of them. Table 5.1 lists the methods that were used and the number of calls made to simulate a product's journey through the supply chain being registered in our system. Each product has five state, location and bearer changes registered, as well as two readings. Four different terms are created for quality checks, and other common methods are used once per product simulation. These were the operations and state changes we thought might serve as an average of typical state changes a product record would go through on our system. The data input sizes varied between separate calls but not between separate product record simulations (meaning the same data was always being introduced with every product record simulation).

Table 5.1: Methods and number of calls made on the simulation of one product's supply chain journey record.

Product Record Simulation	
Method	**Number of Calls**
addState() addLocation() addBearer()	5
createTerm()	4
addReading()	2
addLabel() addCertification() markEOC() fireReadingOrder() fireStopReadingOrder()	1

Figure 5.1 shows the results of the product record simulation[1], with an assumed gas price of 5 Gwei (currently a price that yields quick acceptance time from miners) and ETH market price at the time of simulation [2] used for calculating costs in EUR(€).



√ Sim completed (95966ms, 76807390 gas)

Gas					Block limit: 471238800 gas	
Methods		5 gwei/gas			121.47 eur/eth	
Contract	Method	Min	Max	Avg	# calls	eur (avg)
Product_Labels	createProductGs1	-	-	174937	25	0.11
Product_Manager	addBearer	63920	94668	87987	125	0.05
Product_Manager	addCertification	-	-	76421	25	0.05
Product_Manager	addLocationToPath	99593	116513	103460	125	0.06
Product_Manager	addProductState	56499	86499	71499	125	0.04
Product_Manager	createProduct	277684	322634	281276	25	0.17
Product_Manager	createTerm	173488	188730	184831	100	0.11
Product_Manager	fireReadingOrder	-	-	40577	25	0.02
Product_Manager	fireStopOrder	-	-	40731	25	0.02
Product_Manager	markEndOfChain	-	-	63985	25	0.04
Product_Manager	newDevice	-	-	107010	1	0.06
Product_Manager	registerReading	145950	161448	153699	50	0.09
Product_Manager	registrarCreateUser	142096	144885	143186	5	0.09
Deployments					% of limit	
Product_Labels		-	-	515363	0.1 %	0.31
Product_Manager		-	-	17928724	3.8 %	10.89

1 passing (10m)

Figure 5.1: Gas cost results on the simulation of 25 product tracking supply chain processes. Obtained with eth-gas-reporter[67].

The test results seen here display the average gas cost of every method called in the simulation, the gas cost of contract deployment, and the equivalent cost in EUR(€), putting in evidence just how costly launching and using the smart contract system on the public Ethereum network would be. It should be taken into account that variations on gas cost between calls are quite hard to predict, and we tried to keep all variable data size insertions on a similar level. The maximum, minimum and average gas costs for each method do not change with an increase in number of product tracking data simulations because the input data and initial state are always the same.

Notably, the deployment of the main Product Manager smart contract is the most expensive call undertaken in our test, costing 17,928,724 gas. As we have mentioned

[1] some of the method names listed in this figure are slightly longer versions of names used in implementation descriptions, such as on Section 4.2.3.

[2] on 23/03/19, at around 23.00H (GMT).

before, a large code size lead to the contract creation surpassing normal block gas cost limits - Ethereum currently has a block gas limit of approximately 8,000,000 gas, which is less than half the value our contract needs. The block gas limit was increased in our test blockchain so deployment of the contract could happen.

Comparatively to deployment costs, calling methods that execute state changes in the smart contract is fairly cheap. The total average gas and EUR(€) cost of calling any method on our smart contract system is 117,662 gas and 0.07€. The EUR(€) cost of transactions is definitely quite high compared to normal, centralized database solutions. It should be kept in mind that we planned for our system to function in a private blockchain, and that the gas limits and prices found here could eventually be managed to be much lower than the ones shown, which are relative to a specific public blockchain.

Furthermore, taking into account the Ethereum block gas limit of 8 million gas, we could fit on average 68 smart contract transactions into a normal Ethereum block. Since Ethereum blocks are mined approximately at every 10s, under normal Ethereum blockchain network configurations the resulting transaction throughput on a private blockchain - 408 transactions per minute - should satisfy a high-scale solution's needs.

5.1.3 State Storage Scalability

The last analysis we made of our smart contract system was focused on storage costs. For an increasing number of consequent product record simulations such as the one defined in Table 5.1, we used ganache-cli to store the blockchain state database being created in a folder and upon the end of the simulations, retrieved the folder's size values. The results, plotted with Matlab, can be seen in Figure 5.2, showing the variation of the blockchain's state database size - actual size, $s_{storage}$, (in blue) and size on disk , $ds_{storage}$,(in red), in megabytes.

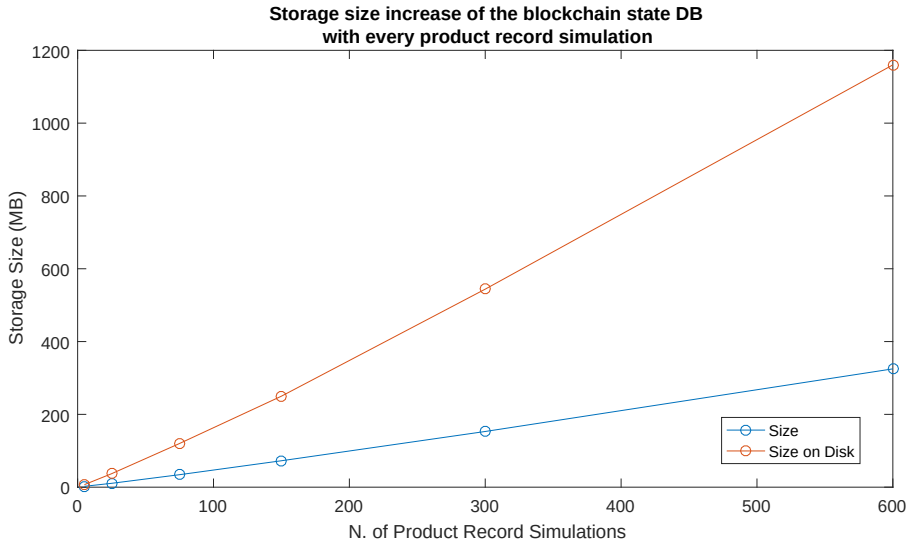


Figure 5.2: Plot showing the blockchain's state database size increases (actual size and size on disk) versus the number of product record simulations.

The results show that within a range of 5 to 600 product record simulations, the size increase of the state database is approximately linear. Taking into account that the product records being simulated have a constant size, these results fall within the expected. To better view the data storage scalability of the system without the need for extremely lengthy simulations, we use the results of the product record simulations we have done to create two first order models, one for storage size and another for storage size on disk, that replicate the approximate behaviour of storage increases in our blockchain's state database with n being the number of product record simulations. The equations obtained with the `polyfit()` command in Matlab were the following:

$$s_{storage}(n) = 0.5442 \times n - 5.1044 \tag{5.1}$$

$$ds_{storage}(n) = 1.9457 \times n - 21.5432 \tag{5.2}$$

With these models we again plot a graphic showing expected size and size on disk storage being occupied by the Ethereum state, but this time for a maximum of 120,000 simulations of a product record. This can be seen in Figure 5.3.

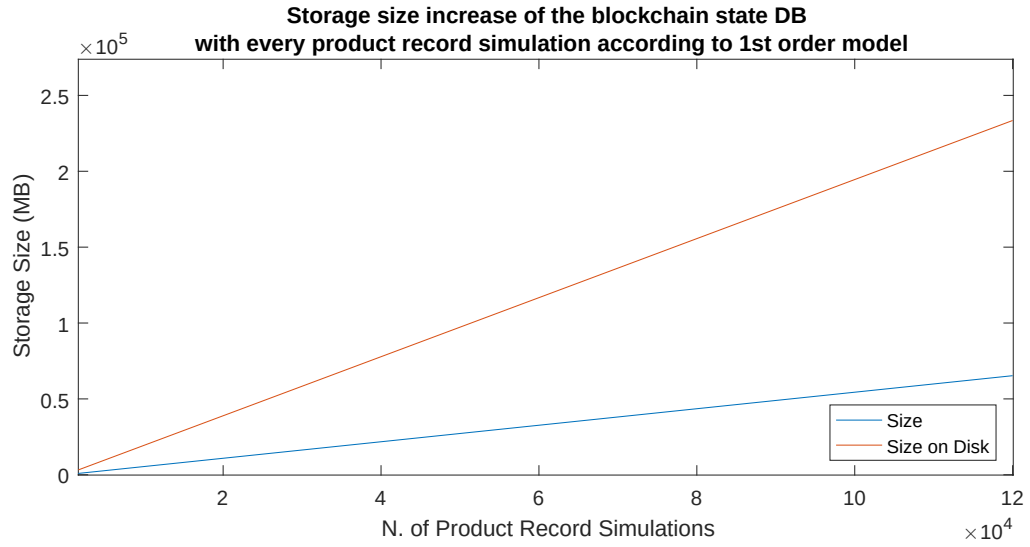


Figure 5.3: Plot made with the first order models of Equations 5.1 and 5.2, showing the blockchain's state database size increases versus the number of product record simulations.

The values of the state database's files actual size versus the size on disk show that the database is being stored on disk in a highly inefficient way. This is likely due to the fact that ganache-cli mines a block for every transaction, so the number of files created in the storage database - comprising of blocks and transactions- is probably much higher than what would happen in a full Ethereum blockchain node, resulting in the bloating of the total storage space spent on disk that is seen here. In a real blockchain node with proper mining, the disk space occupied by the state database should be much closer to the actual data size values than what is seen in these results.

Nonetheless, according to these estimates, and assuming that our blockchain system is being used by a number of different companies to track their products, after a year of usage where the average number of product's tracked per month is 10,000, then the 120,000 product records existent in the system would be kept in a state database with 65.3 GB in size, occupying up to 232.4GB of disk space. After another year with the same usage rates, these values would double. Considering only the actual state database size, the storage values are certainly possible to sustain with modern computers.

If we consider the unsustainable to be say, 1TB of data (size, not on disk), then the limit would be reached at around 1.837 million product records. At the usage rates we mentioned of 120,000 records per year, the 1TB mark would be hit after 15 years of system usage. These values are hard to put into perspective if we don't know how to answer two essential questions relative to system size through time:

1. What system usage scale do we want to achieve? How many records are being created per month?

2. How long do we want to keep these records stored on the blockchain? Is there an expiry date for the usefulness of this data?

To answer the first question, we attempted to find some statistics on logistics parcels that could be of use. We found one claiming that in 2016, 65 billion parcels in total were shipped from one country to others [68]. This means that if a 1TB state database size limit were established for our blockchain system, yearly it could handle tracking data on $2.82 \times 10^{-3}\%$ of total worldwide parcel shipping. The perspective this gives us is that our blockchain system could maybe handle regional levels of tracking data, but could probably not be applied to say, the scale of a small country.

The second question might have an answer if we consider that many products are perishable, and after two years of being processed, it is unlikely that they are still on sale. If 1TB is the size limit in a two year window, then 1.837 million product records could be created during those two years, at an average of 76,541 products being processed per month. After this set two year period, the blockchain managers could delete product records that are unused and stabilize storage levels.

Because we found a limited number of statistics to put these values into perspective, it would be up to logistics companies to decide if metrics like these are up to par with their businesses. The overall conclusion on scalability is that the system could not be easily applied to truly large-scale solutions in large part due to state storage constraints.

5.2 Browser Applications

Two simple browser applications were built in order to validate our application proposal. One is meant for regular users who have bearer permissions and want to access information on the blockchain. The second application serves as a bridge to communication between an Arduino UNO microcontroller connected to a temperature sensor and our smart contract system.

5.2.1 Product Tracking Viewer

In Section 4.3 an overview of the tools used to develop our web applications was given. A screenshot of the product tracking viewer app we developed is shown in Figure 5.4.

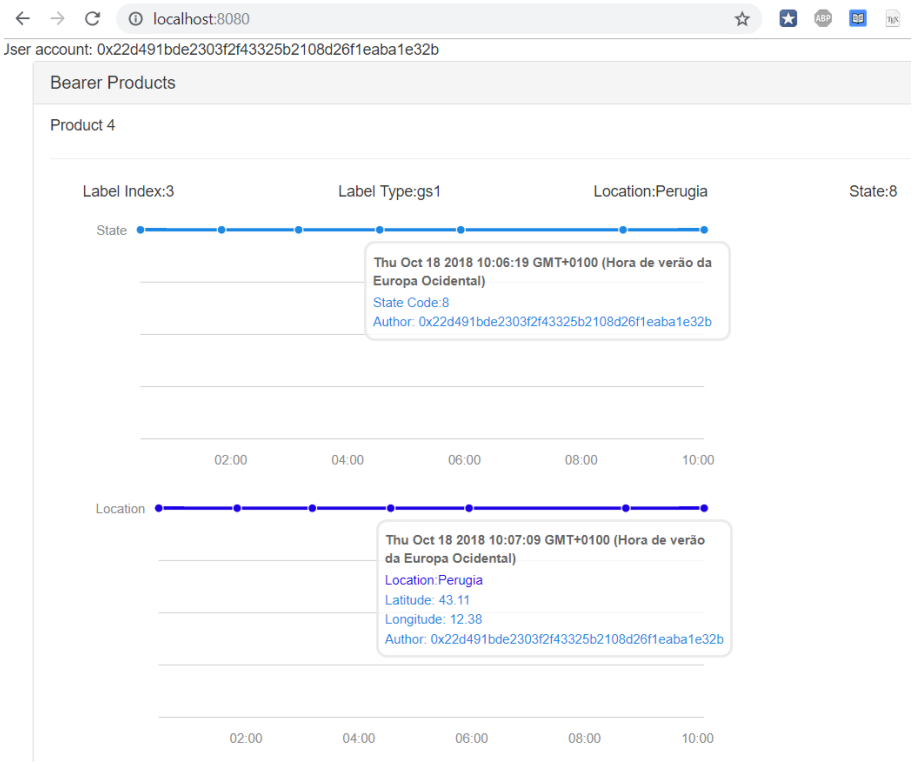


Figure 5.4: Screenshot of the product tracking viewer application in use.

The way the application functions is the following:

1. The user logs into his account using the Metamask browser add-on, and connects to the blockchain network where the smart contract system is deployed.

2. When the user accesses the application's web page, Metamask provides the application with a connection to the blockchain where the user is logged into (which should be the blockchain where the contract has been deployed).

3. The web application then has all the tools needed (user address, contract address and contract interface) to query the blockchain for a list of items that the user is

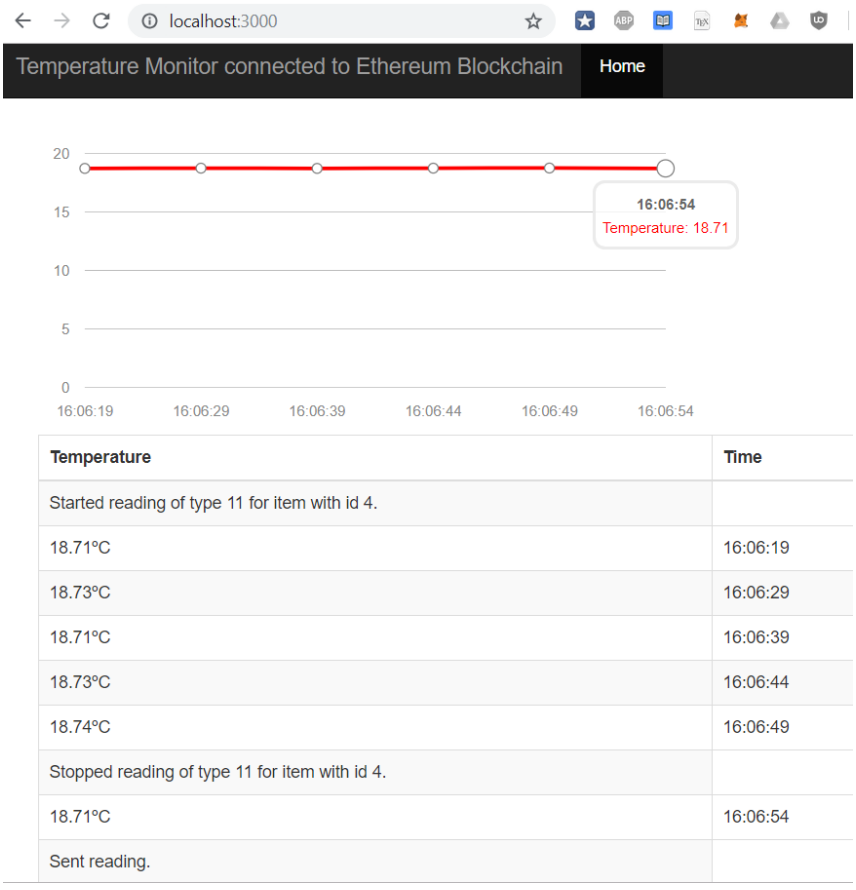
currently bearer of. It takes the ID numbers retrieved to query information on each product - label data and all tracking data.

4. A list of drop-down items is created for each product ID retrieved. Upon opening a drop-down menu item, it displays that product's tracking data changes in a timeline graphic - namely state changes and location changes (see Figure 5.4.)

In this way the application can be used to keep track of state and location changes of items. The methods shown here could eventually be used to expand the functionalities of the web application to data insertion and more viewing options (such as on clearance checks).

5.2.2 Temperature Monitor and Blockchain Connection Application

The second web application we built manages communication between a temperature sensor connected to an Arduino microcontroller and our Etherem blockchain. Its function and implementation were already discussed in detail in Section 4.4. A screenshot of the final application is shown in Figure 5.5, where temperature readings, order receivals and the sending of a reading were registered.



Temperature	Time
Started reading of type 11 for item with id 4.	
18.71°C	16:06:19
18.73°C	16:06:29
18.71°C	16:06:39
18.73°C	16:06:44
18.74°C	16:06:49
Stopped reading of type 11 for item with id 4.	
18.71°C	16:06:54
Sent reading.	

Figure 5.5: Screenshot of the blockchain to IoT bridge application being used.

Summarizing what was described in Section 4.4, our second web application connects to the blokchain via Metamask, a browser add-on, on which a person has logged in the device's account and then connected to the blockchain, where our smart contracts have been deployed.

Our web application, upon establishing communication with the blockchain, monitors the launch of new blocks for events that order the start or end of temperature readings for the specific device that is logged into Metamask. If such an order is received from the device's owner, the application sends to the blockchain an event acknowledging the successful order transmission, and starts collecting temperature readings of the type requested.

Upon receiving an event order to stop reading values, the application collects the results of the reading and registers them to the blockchain on the appropriate Product Record with the device's account. It should be noted the application was also built with the management of several orders at the same time in mind.

The only difficulty found in this implementation result is that by using Metamask as a wallet, there was the inconvenient of having to confirm every transaction made to the blockchain, which we would not want to find on an application that is automatized. This is because wallets are obviously meant to ensure every transaction they do has been approved by the user and has been done safely, but for the purpose we have here, and taking into account the number of safety/authentication precautions on the smart contract side, it is not ideal to require user interaction. Regardless, the order system was designed so that reading data would not be affected by delays on user approval, so the system works correctly. An improved implementation should be based on the use of a different/custom wallet software being used.

Chapter 6

Conclusion

This dissertation has presented the design and implementation of a blockchain smart contract system geared towards management of products in a logistics system. To this end, a number of different features have successfully been implemented in a decentralized blockchain system, namely:

1. a Role-Based Access Control (RBAC) system, implemented with smart contract technologies;
2. a decentralized registry for tracking and tracing data of products in supply chains;
3. automated clearance and quality control feautures that take into account IoT device input;
4. a web application that accesses the smart contract's tracking data and displays it for users;
5. a web application that serves as a communication bridge between an IoT device (an Arduino UNO microcontroller connected to a temperature sensor) and the blockchain smart contract system.

The solution presented here showcases the potential of blockchain technology in solving some of the issues facing the logistics industry, whilst taking advantage of the decentralized character of these systems. Transparency is achieved by making the tracking data available to all logistics entities and customers. Since the system is decentralized, there are no mediators in control of it, and data integrity is kept on a trusted, shared database.

The smart contract system developed in this work passed all tests made on its implemented features. Further tests revealed that although the code size of the main smart contract had given issues in terms of breaking network size and cost limits upon deployment, the methods that were built into it do not suffer from the same issues and have

manageable costs. We concluded that in a private blockchain configured similarly to the Ethereum public blockchain, an average throughput of 408 transactions per minute with our smart contract could be achieved.

Our tests on storage size scalability have shown that again, in a private blockchain, the system can potentially be used to store the tracking data of up to 1.837 million products in 1TB of storage. Compared to number of parcels being processed yearly in the world - 65 billion[68] - the scalability of our solution is very limited by storage size constraints and could possibly only be used at a regional scale of operations. This conclusion would benefit from more statistics to corroborate it, however, which we were unable to find.

Regarding the expected time of operation, we find that for the system to be usable for large periods of time by a considerable number of companies/users, a periodic deletion of state storage data might have to be done. This decision might put at stake the benefits of transparency and untamperability that blockchain brings, but it is an option to consider, since the relevancy of tracking data decreases with its ageing.

Integration with IoT devices was one of the points made difficult due to the processing power requirements of blockchain clients and an overall shortage of dedicated tools and resources to this end. However we have found that the connection and integration of web applications and Ethereum blockchains is very well supported an has great potential, provided the developers of these applications are well versed in web development.

Many improvements to blockchain technologies are being made towards fixing these issues, and at a very fast pace - so fast in fact that throughout the six months this work's implementation was developed, some technologies we used are already becoming outdated. It is consequently our belief that the viability and usability of systems such as the one we have presented here will only become more real in the near future, and may well revolutionize the transparency for supply chains as we know them.

Overall, we believe that the smart contract system developed in this work, as well as the applications built to interact with it, are valid proof of the feasibility of blockchain solutions for supply chains. Even with the limitations we found the technologies have great potential to bring decentralization and transparency to supply chain processes.

6.1 Future Work

In future work, we would like to fix the smart contract code size issue we found and implement communication of the smart contracts seen in Section 4.2 via interfaces, so that they could be deployed separately and maybe eventually be used in networks that are restrictive in regards to contract code and block sizes, such as the main public Ethereum blockhain.

Another point of interest would be developing a private Ethereum blockchain with characteristics tailored to the application developed here - such as higher gas limits, a consensus system more adapted to consortium needs and dedicated to running transactions on our smart contract system only.

We would also like to implement the communication of more types of IoT sensors with our blockchain (i.e. RFID identifiers), and explore the benefits of their usage in supply-chain processes by testing our system in real use cases.

Finally, we would like to obtain more statistics on supply chain processes and logistics. These could be used to make a more thorough cost analysis of our system and eventually find improvements to it that would better fit the industry's needs.

6.2 Scientific Contributions

The work developed here resulted in the writing of two scientific articles, submitted to the following conferences:

- L. Augusto, R. Costa, J. Ferreira, R. Jardim-Gonçalves, "An Application of Ethereum Smart Contracts and IoT to logistics" YEF-ECE 2019 - 3rd International Young Engineers Forum on Electrical and Computer Engineering, Caparica. Paper was accepted and presented.
- L. Augusto, R. Costa, J. Ferreira, R. Jardim-Gonçalves, "An application of Ethereum smart contracts to logistics", SERVICES 2019 - World Congress on Services, San Diego. Paper was accepted.

Bibliography

[1] Aaron Wright and Primavera De Filippi. "Decentralized blockchain technology and the rise of lex cryptographia." In: (2015).

[2] Marco Iansiti and Karim R Lakhani. "The truth about blockchain." In: *Harvard Business Review* 95.1 (2017), pp. 118–127.

[3] Satoshi Nakamoto. "Bitcoin: A peer-to-peer electronic cash system." In: (2008).

[4] R. Beck. "Beyond Bitcoin: The Rise of Blockchain World." In: *Computer* 51.2 (2018), pp. 54–58. ISSN: 0018-9162. DOI: 10.1109/MC.2018.1451660.

[5] Sloane Brakeville and Bhargav Perepa. *Blockchain basics: Introduction to distributed ledgers*. (Accessed on 08/20/2018). 2018. URL: https://www.ibm.com/developerworks/cloud/library/cl-blockchain-basics-intro-bluemix-trs/index.html.

[6] IBM Institute for Business Value. *Trust in trade: Toward Stronger Supply Chains*. (Accessed on 08/10/2018). 2016. URL: https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=GBE03771USEN.

[7] CK Panetta. *Top trends in the Gartner Hype Cycle for emerging technologies. 2017*. (Accessed on 08/01/2018). 2017. URL: https://www.gartner.com/smarterwithgartner/top-trends-in-the-gartner-hype-cycle-for-emerging-technologies-2017/.

[8] Gavin Wood. "Ethereum: A secure decentralised generalised transaction ledger." In: *Ethereum project yellow paper* 151 (2014), pp. 1–32.

[9] Michael Crosby, Pradan Pattanayak, Sanjeev Verma, and Vignesh Kalyanaraman. "Blockchain technology: Beyond bitcoin." In: *Applied Innovation* 2 (2016), pp. 6–10.

[10] Andrew Allen. *Logistics industry to be worth 15.5tn by 2023*. (Accessed on 06/27/2018). 2016. URL: https://www.cips.org/supply-management/news/2016/november/logistics-industry-forecast-to-be-worth-155tn-by-2023/.

[11] Alan Punter. "Supply Chain Failures: A study of the nature, causes and complexity of supply chain disruptions." In: *Airmic* (2013).

[12] Savills Investment Management. *European Logistics - Warehousing the Future*. (Accessed on 08/03/2018). 2017. URL: http://www.savillsim.com/documents/2017-sim-european-logistics-warehousing-the-future-final.pdf.

[13] Gina Chung Markus Kückelhaus. *Blockchain in Logistics*. (Accessed on 08/27/2018). 2018. URL: https://www.logistics.dhl/content/dam/dhl/global/core/documents/pdf/glo-core-blockchain-trend-report.pdf.

[14] Mitsuaki Nakasumi. "Information Sharing for Supply Chain Management based on Block Chain Technology." In: *2017 IEEE 19th Conference on Business Informatics* (2017).

[15] Olivia Krauth. *5 companies using blockchain to drive their supply chain*. (Accessed on 08/03/2018). 2018. URL: https://www.techrepublic.com/article/5-companies-using-blockchain-to-drive-their-supply-chain/.

[16] *Blockchain: the solution for supply chain transparency | Provenance*. https://www.provenance.org/whitepaper. (Accessed on 02/17/2019). 2015.

[17] Torsten Stein. *Supply chain with blockchain—showcase RFID*. (Accessed on 08/10/2018). 2017.

[18] Blockgeeks. *What Is Hashing? Under The Hood Of Blockchain*. (Accessed on 08/19/2018). 2017. URL: https://blockgeeks.com/guides/what-is-hashing/.

[19] Kiran Vaidya. *Bitcoin's implementation of Blockchain*. (Accessed on 08/17/2018). 2016. URL: https://medium.com/all-things-ledger/bitcoins-implementation-of-blockchain-2be713f662c2.

[21] F. Tschorsch and B. Scheuermann. "Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies." In: *IEEE Communications Surveys Tutorials* 18.3 (2016), pp. 2084–2123. ISSN: 1553-877X. DOI: 10.1109/COMST.2016.2535718.

[22] Kiran Vaidya. *Decoding the enigma of Bitcoin Mining Part I : Mechanism*. (Accessed on 08/18/2018). 2016. URL: https://medium.com/all-things-ledger/decoding-the-enigma-of-bitcoin-mining-f8b2697bc4e2.

[23] Vitalik Buterin et al. "A next-generation smart contract and decentralized application platform." In: *white paper* (2014).

[24] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. "An overview of blockchain technology: Architecture, consensus, and future trends." In: *Big Data (BigData Congress), 2017 IEEE International Congress on*. IEEE. 2017, pp. 557–564.

[25] *Quorum | J.P. Morgan*. https://www.jpmorgan.com/global/Quorum. (Accessed on 02/03/2019).

[26] *Hyperledger Fabric Introduction*. (Accessed on 06/20/2018). URL: https://hyperledger-fabric.readthedocs.io/en/release-1.2/whatis.html.

[27] Tien Tuan Anh Dinh, Rui Liu, Meihui Zhang, Gang Chen, Beng Chin Ooi, and Ji Wang. "Untangling Blockchain: A Data Processing View of Blockchain Systems." In: *CoRR* abs/1708.05665 (2017). arXiv: 1708.05665. URL: http://arxiv.org/abs/1708.05665.

[28] Mic Bowman Kelly Olson et al. *Sawtooth:An Introudction*. (Accessed on 06/20/2018). 2018. URL: https://www.hyperledger.org/wp-content/uploads/2018/01/Hyperledger_Sawtooth_WhitePaper.pdf.

[29] *GitHub - hyperledger/burrow: Hyperledger Burrow*. (Accessed on 06/20/2018). URL: https://github.com/hyperledger/burrow.

[30] *Consortium Chain Development ethereum/wiki*. https://github.com/ethereum/wiki/wiki/Consortium-Chain-Development. (Accessed on 02/03/2019).

[31] Nick Szabo. *The Idea of Smart Contracts*. (Accessed on 06/18/2018). 1997. URL: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/idea.html.

[32] Vitalik Buterin. "Ethereum: Platform Review." In: *Opportunities and Challenges for Private and Consortium Blockchains* (2016).

[33] *Solidity in Depth — Solidity 0.5.3 documentation*. https://solidity.readthedocs.io/en/v0.5.3/solidity-in-depth.html.

[34] *Get Started OpenZeppelin*. https://openzeppelin.org/api/docs/get-started.html. (Accessed on 12/20/2018).

[35] *ETH Gas Station | Consumer oriented metrics for the Ethereum gas market*. https://ethgasstation.info/index.php. (Accessed on 12/16/2018).

[36] *$86.56 (ETH/USD) EthereumPrice.org - USD Price, Charts & History*. https://ethereumprice.org/. (Accessed on 12/16/2018).

[37] Val Srinivas Jesus Leal Trujillo Steve Fromhart. "Evolution of blockchain technology." In: (2017).

[38] Collin Sullender. *Scholar Plot*. 2018. URL: https://www.csullender.com/scholar/.

[39] Y. Yuan and F. Wang. "Towards blockchain-based intelligent transportation systems." In: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. 2016, pp. 2663–2668. DOI: 10.1109/ITSC.2016.7795984.

[40] L. Xu, L. Chen, Z. Gao, Y. Lu, and W. Shi. "CoC: Secure Supply Chain Management System Based on Public Ledger." In: *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. 2017, pp. 21–6. DOI: 10.1109/ICCCN.2017.80385142.

[41] Feng Tian. "An agri-food supply chain traceability system for China based on RFID blockchain technology." In: *2016 13th International Conference on Service Systems and Service Management (ICSSSM)*. 2016, pp. 1–6. DOI: 10.1109/ICSSSM.2016.7538424.

[42] "Supply Chain Object Discovery with Semantic-enhanced Blockchain." In: 2017. ISBN: 9781450354592. DOI: 10.1145/3131672.3136974.

[43] *Maersk and IBM Introduce TradeLens Blockchain Solution*. http://newsroom.ibm.com/2018-08-09-Maersk-and-IBM-Introduce-TradeLens-Blockchain-Shipping-Solution. (Accessed on 08/9/2018).

[44] *GitHub - hyperledger/sawtooth-supply-chain: Hyperledger Sawtooth Supply Chain*. (Accessed on 06/20/2018). URL: https://github.com/hyperledger/sawtooth-supply-chain.

[45] *FishNet*. (Accessed on 08/30/2018). URL: https://demo.bitwise.io/fish/#!/.

[46] *The Latest Blockchain Supply Chain Startups – Medium*. https://medium.com/applicature/the-latest-blockchain-supply-chain-startups-7b0e5c07548f. (Accessed on 02/16/2019).

[47] *WALTONCHAIN*. URL: https://www.waltonchain.org/.

[48] *Emergent Technology™ And Yamana Gold To Improve Gold Supply Chain*. (Accessed on 09/15/2018). URL: https://www.prnewswire.com/news-releases/emergent-technology-and-yamana-gold-to-improve-gold-supply-chain-300596816.html.

[49] *Wine Blockchain | The digital traceability and certification of wine | Ez Lab*. (Accessed on 09/12/2018). URL: http://www.ezlab.it/case-studies/wine-blockchain/.

[50] *GitHub - imperialsoftech/coffee-supplychain-ethereum: Implementation of coffee supplychain using ethereum smart contract addressing the issue of storing critical data necessary at different stages of supplychain and making it verifiable by all stakeholders in supplychain*. (Accessed on 09/12/2018). URL: https://github.com/imperialsoftech/coffee-supplychain-ethereum.

[51] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. "Internet of Things (IoT): A vision, architectural elements, and future directions." In: *Future generation computer systems* 29.7 (2013), pp. 1645–1660.

[52] Konstantinos Christidis and Michael Devetsikiotis. "Blockchains and smart contracts for the internet of things." In: *IEEE Access* 4 (2016), pp. 2292–2303.

[53] Nir Kshetri. "Can blockchain strengthen the internet of things?" In: *IT Professional* 19.4 (2017), pp. 68–72.

[54] Matevž Pustišek and Andrej Kos. "Approaches to Front-End IoT Application Development for the Ethereum Blockchain." In: *Procedia Computer Science* 129 (2018), pp. 410–419.

[55] *Trusted IoT Alliance*. (Accessed on 08/16/2018). URL: https://www.trusted-iot.org/.

[56] Mathias Davidsen, Sebastian Gajek, Marvin Kruse, and Sascha Thomsen. "Empowering the Economy of Things." In: (2017). URL: https://weeve.network/cache/assets/fo0rvac1gv8v/5Ui5mhVJKgGqYqe6O2OESm/a353cd67e2e9398ad552771fc09187d5/Weeve_Technical_Whitepaper.pdf.

[57] *Trade And Transport Status Codes*. https://www.unece.org/fileadmin/DAM/cefact/recommendations/rec24/rec24_ecetrd258e.pdf. (Accessed on 01/27/2019).

[58] *GS1 Logistics Label Guideline*. https://www.gs1.org/docs/tl/GS1_Logistic_Label_Guideline.pdf. (Accessed on 03/21/2019).

[59] *GitHub - trufflesuite/truffle: The most popular blockchain development framework*. (Accessed on 02/01/2019). URL: https://github.com/trufflesuite/truffle.

[60] *GitHub - trufflesuite/ganache-cli: Fast Ethereum RPC client for testing and development*. (Accessed on 02/01/2019). URL: https://github.com/trufflesuite/ganache-cli.

[61] *GitHub - ethereum/web3.js: Ethereum JavaScript API*. (Accessed on 02/01/2019). URL: https://github.com/ethereum/web3.js/.

[62] Andreas M Antonopoulos and Gavin Wood. *Mastering ethereum: building smart contracts and dapps*. O'Reilly Media, 2018.

[63] *Arduino Code | Adafruit SHT31-D Temperature & Humidity Sensor Breakout | Adafruit Learning System*. https://learn.adafruit.com/adafruit-sht31-d-temperature-and-humidity-sensor-breakout/wiring-and-test. (Accessed on 03/22/2019).

[64] *GitHub - firmata/protocol: Documentation of the Firmata protocol*. https://github.com/firmata/protocol. (Accessed on 03/04/2019).

[65] *Socket.IO*. https://socket.io/. (Accessed on 03/04/2019).

[66] *dweet.io - Share your thing*. https://dweet.io/. (Accessed on 03/04/2019).

[67] *eth-gas-reporter - npm*. https://www.npmjs.com/package/eth-gas-reporter. (Accessed on 03/24/2019).

[68] • *Chart: 65 Billion Parcels Were Shipped in 2016 | Statista*. https://www.statista.com/chart/10922/parcel-shipping-volume-and-parcel-spend-in-selected-countries/. (Accessed on 03/25/2019).