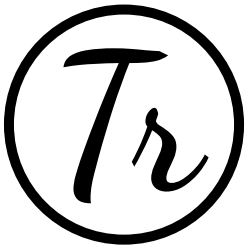
TÉCNICO LISBOA



Blockchain for Supply Chain Traceability and Governance in the Agri-food Sector

Duarte Miguel Constantino Bento

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisors: Prof. Miguel Ângelo Marques de Matos
Prof. Miguel Nuno Dias Alves Pupo Correia

Examination Committee

Chairperson: Prof. António Paulo Teles de Menezes Correia Leitão
Supervisor: Prof. Miguel Ângelo Marques de Matos
Member of the Committee: Prof. Vinícius Vielmo Cogo

November 2023

Declaration
I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

I want to start by thanking my family, especially my parents Ana and Luís for their perseverance in instilling strong values and their continuous support over this 5-year journey. To my sister, Ana, for her unconditional love and support. And to Susana, my girlfriend, for her invaluable support that motivates me to move forward, and especially for sharing some of her creativity and insights with me.

I would also like to express my gratitude to my advisors Prof. Miguel Matos and Prof. Miguel Correia for the insight, guidance, and encouragement that was instrumental in completing this thesis.

To the friends I made during my academic journey. To my fellow firefighters that teased but always believed. To all that were part of my life for these last 5 years, for the experiences we shared and the moments that shaped me into the person I am today – Thank you.

In memory of my grandmother, Graciete, who unfortunately could not see me finish this chapter of my life, but whom I will eternally respect.

This work was partially supported by the European Union under the project grant TrustyFood (EU H2020 101060534).

Abstract

Supply chains, notably in the agri-food industry, face a growing need for transparency, traceability, and improved food safety as consumer awareness and sustainability needs increase. To address these challenges, we present Tracer, a blockchain-based supply chain traceability system. Leveraging Ethereum's smart contracts, Tracer provides a transparent, secure, and decentralized ledger of transactions recorded throughout the supply chain. It also incorporates an external file system to record large artifacts of product information, e.g. images, aiding consumers in making a more informed purchase decision. The reliability of our system makes for efficient product identification in the event of a food safety incident.

To further secure and increase supply chain trust, our proposal introduces a Decentralized Autonomous Organization (DAO) as the regulatory body of our system. This DAO allows for decentralized decision-making while maintaining regulatory oversight, setting it apart from other traceability systems. Moreover, the system offers user-friendly interfaces, including a consumer-oriented timeline design accessed via QR codes on product packaging.

Recognizing the importance of cost optimization in supply chain technologies, we conducted a comprehensive analysis to refine the system's key aspects. By implementing efficient storage via Ethereum events and off-chain document URIs, we achieved cost-effective data storage. Additionally, we integrated an access control mechanism for our off-chain storage system (IPFS) to safeguard against potential resource exploitation. Finally, we achieved a 71% reduction in the deployment cost of our DAO by tailoring the conventional OpenZeppelin implementation of governance contracts to our specific needs.

Keywords

Blockchain; Ethereum; Supply chain; Agri-food; Traceability; Decentralized Autonomous Organization.

Resumo

As cadeias de abastecimento, especialmente na indústria agro-alimentar, enfrentam uma crescente necessidade de transparência, rastreabilidade e melhorias na segurança alimentar. Neste trabalho, apresentamos o Tracer, um sistema de rastreabilidade da cadeia logística baseado em blockchain. O Tracer fornece um registro transparente, seguro e descentralizado das transações em toda a cadeia de abastecimento, através do uso de ”smart contracts” da plataforma Ethereum. O sistema incorpora também um serviço de armazenamento externo para registrar informações adicionais sobre produtos, com o objetivo de ajudar os consumidores a tomar decisões mais informadas no momento de compra. A confiabilidade do nosso sistema possibilita a identificação eficiente de produtos no caso de um incidente de segurança alimentar. Propomos também uma Organização Autônoma Descentralizada (DAO) como órgão regulador do nosso sistema, permitindo tomar decisões de forma descentralizada, um aspeto que diferencia o nosso sistema de outros na área.

Conduzimos uma análise adicional com foco em otimizar os custos associados ao sistema. Resultou uma redução de 71% no custo de instalação dos “smart contracts” que compõem o nosso DAO, comparativamente com a implementação típica do OpenZeppelin. E para além do uso de um sistema de armazenamento externo, melhoramos ainda o custo de disseminação da informação adicional através do uso de eventos da plataforma Ethereum, e integramos um mecanismo de controle de acesso ao nosso sistema de armazenamento externo (IPFS) para proteção contra a possível exploração de recursos.

Palavras Chave

Blockchain; Ethereum; Cadeia logística; Agro-alimentar; Rastreabilidade; Organização Autônoma Descentralizada.

Contents

List of Figures

List of Tables

Listings

Acronyms

API	Application Programming Interface
RPC	Remote Procedure Call
EVM	Ethereum Virtual Machine
IPFS	InterPlanetary File System
CID	Content Identifier
URI	Uniform Resource Identifier
ABI	Application Binary Interface
DAO	Decentralized Autonomous Organization
EIP	Ethereum Improvement Proposals

1

Introduction

Contents

The agri-food sector has always been of extreme importance but now, with the predicted increase in population to 8.5 billion by 2030 [1], it is becoming more and more important to improve the sustainability of this sector. Recent studies show that the concern and awareness regarding food products are rising among consumers [2] as they become more interested in the origins of products, production methods, genetically modified food, and others [3, 4].

These concerns have been rightfully increasing over the years with several scandals and cases of food adulteration. Cases such as meat contaminated with the “mad cow” disease in the United Kingdom or horse meat being used as a replacement for beef in Europe 2013 [5, 6] have forced governments worldwide to impose new regulations for the food industries [4].

Moreover, according to recent data from the Centers for Disease Control and Prevention [7], in the U.S., around 48 million people get sick, 128,000 are hospitalized, and 3,000 die each year from foodborne illnesses. To respond to this pressing issue, the U.S. Food and Drug Administration (FDA),

defined a list of products that demand traceable information to be registered and set the limit compliance data by 2026.[1]

As a result, food traceability across the supply chain has gathered the attention of both researchers and industries and is considered an important tool to restore and increase consumers' confidence in food safety [4].

In a traditional food supply chain, communication and cooperation between stakeholders are limited to adjacent participants, which reflects on how much of a time-consuming activity it is to track product movements. The traditional supply chain relies on paper-based systems and each participant has to record his own version of events [8]. In case of a food contamination outbreak, traditionally, the investigating agency would need to request records from every potential participant in that product's supply chain. A transparent traceability system would facilitate the tracking of the affected products for a faster recall.

In this document, we present Tracer, a system focused on supply chain traceability that leverages blockchain technology to tackle some of the challenges of modern supply chains.

1.1 Blockchain

Blockchain is an emerging technology that is gaining popularity and maturity, with a growing number of use cases in areas such as decentralized finance (DeFi), healthcare, and supply chain management [9]. A blockchain is a decentralized, distributed ledger that records transactions across several computers. It uses cryptography to ensure the security and immutability of the recorded data and consensus algorithms to distribute blocks (multiple transactions) across system nodes, maintaining a common true version of the blockchain.

One important blockchain concept for our context is smart contracts. These are programs that autonomously execute as the result of a transaction sent to the blockchain network. The actions performed by these smart contracts are predetermined and embedded in the code deployed to the platform beforehand.

Our approach employs blockchain technology with smart contracts capabilities to improve most supply chains, with a special focus on the agri-food sector, leading to increased consumer confidence in the products being consumed. This system will enable more efficient communication and cooperation between stakeholders, without the need for a centralized authority. Instead, we propose a Decentralized Autonomous Organization (DAO) to oversee and regulate our supply chain traceability system, an approach that, to the best of our knowledge, has not been proposed by any other work. To support this assertion, we conducted an extensive web search and queried the "Scopus" database, one of the largest

[1] https://www.fda.gov/food/food-safety-modernization-act-fsma/fsma-final-rule-requirements-additional-traceability-records-certain-foods

repositories of blockchain publications [10]. We used the following keywords: “DAO”, “Supply chain”, and “Blockchain”.

1.2 Objectives

The objectives and expected contributions of this work are:

- Develop a framework for supply chain traceability that supports agri-food products.
- Conceptualize and implement a DAO for supply chain management.
- Deploy a prototype and evaluate the effectiveness of the proposed system.

1.3 Overview

The remainder of the document is structured as follows. Chapter 2 details the agro-food supply chain and its demands, includes a brief introduction to blockchain technology, and discusses relevant works in agri-food supply chain traceability. Chapter 3 explains our proposed traceability system, including the interactions between core concepts and implementation-specific decisions. Chapter 4 evaluates our system and the design decisions taken to improve the overall system efficiency. Finally, Chapter 5 concludes the document with some final remarks, a short discussion, and possible future work.

2

Related Work

Contents

In this chapter, we will first go through the necessary background to better understand the agri-food supply chain (Section 2.1), including consumer requirements. Secondly, we present a short overview of blockchain technology focusing on Ethereum and Hyperledger Fabric (Section 2.2). Finally, after the relevant background is provided, we analyze some of the available works that propose the use of blockchain technology in the agri-food supply chain (Section 2.3).

2.1 Agri-food Supply Chain

The increasing consumer awareness about sustainability needs and the growing necessity for food traceability has been made evidently important with recent cases of food fraud and incidents such as the horse meat scandal [5, 6]. A traceability system that establishes a reliable chain of custody serves various purposes, such as helping to assess a product's provenance and address food safety issues, benefiting

both supply chain stakeholders and customers.

A classic use-case of food traceability is called from-farm-to-fork and the goal is to provide certified information at every stage of the supply chain, from the producer (farm) up until the consumer (fork) [11].

2.1.1 Actors in Supply Chain

Every supply chain is composed of different actors that somehow participate in the product's journey. For us consumers, it is sometimes difficult to infer the role of every participant. To simplify this, authors usually define a set of classes to encompass the involved actors.

Caro et al. [11] defined actors in the agri-food supply chain into 6 categories:

A) provider: providers of raw materials, such as seeds and nutrients, but also pesticides, chemicals, etc;

B) producer: entity that produces a product;

C) processor: this actor may perform various actions, from simple packaging to more complex processes (e.g., pressing of the olives);

D) distributor: this actor is responsible for moving the output product from the processor's site to retailers;

E) retailer: this actor is responsible for selling the products, representing either small local stores or big supermarkets;

F) consumer: the final element of the chain.

Granillo-Macías and Isidro J. González-Hernández [5] limit the specification to farmers, distributors, processors, wholesalers, retailers, and end consumers, whereas Rana et al. [12] generalize to producers, suppliers, transporters, wholesalers, retailers, other intermediaries, and customers. These classes are attributed general names so that customers can intuitively recognize their contribution to the supply chain. We choose to adopt the model from Caro et al. [11] as it is generic but complete.

2.1.2 Requirements

The agri-food sector is significantly shaped by consumer expectations and requirements when it comes to the food products they buy and consume. Ensuring that these requirements are met is important for building and maintaining customer trust and satisfaction.

In this section, we present an in-depth look at the most relevant requirements in the context of our work [13]:

Traceability

The European Union (EU, 2002) defines traceability as "the ability to trace and follow a food, feed, food-producing animal or substance intended to be, or expected to be incorporated into a food or feed, through all stages of production, processing and distribution" [14].

Traceability can be regarded as recording the flow of a product throughout its lifecycle, ensuring that information about physical movements (*logistics tracking*) of a product is accessible from participant to participant, as described earlier by the from-farm-to-fork scenario [8, 12]. Logistic information is the subproduct of a transaction, for example, the names and addresses of a processor and distributor taking part in an exchange, as well as the date, batch identifier, and volume or quantity[1]. Basic supply chain tracing requires only stakeholder identification and transaction logging.

Traceability is usually considered an important tool to help increase consumer confidence but it often comes associated with higher product prices. Nevertheless, research on consumers' attitudes in China reveals that the increased knowledge positively affected consumers' will to pay for such products, justifying the higher prices [4].

Traceability is the foundation for the following requirements since its correct implementation enables several other ramifications like food authenticity, credibility, and faster deployment of corrective measures like food recall.

Transparency

Although most customers are interested in the journey their products have taken, they also value information regarding the production, storage, and transportation. This is considered *qualitative tracking* [12], where extra information to the already provided in traceability systems is supplied by each actor while they hold the product. This can be production methods recorded by producers, storage conditions by processors, CO_2 emissions by distributors, or any other relevant information.

The European Union regulation on food (EU, 2002) [14] solely requires that businesses include information about the actor with whom they interact directly "to ensure that on investigation, traceability can be assured at all stages". This means traceability documents are only stored at each member, which is enough when it comes to investigations by an authorized organization [12] but cannot be accessed by the general consumer. For that to happen the information needs to be transparently recorded and easily accessible by everyone.

Transparency is becoming increasingly important, as consumers require it as a reassuring measure of quality and safety. Businesses are being pressured into extending transparency into their supply chains but they too can take advantage of this influence [3]. Over 71 percent of consumers around the world are willing to pay a premium for brands that provide transparent information [15]. Stakeholders

[1] `https://www.foodstandards.gov.au/industry/safetystandards/traceability`

can also gain from this transparent information by adjusting their activities according to the available data. For example, a producer that observes an increased demand for one of their products can adjust its production to match the demand without going through the process of requesting retailers' inventory data.

Information Integrity

In order to maintain trust along the supply chain the stored records need to be tamper-proof [11]. The involved actors should not be allowed to modify information previously submitted, otherwise, all the data would be considered unreliable, as they could modify it immediately after the product transitions to the next entity in the supply chain. For this reason, keeping data integrity is crucial to reliably support both traceability and transparency properties.

Analogous to the previous argument, the supply chain should not be administrated by a centralized authority with control over all information. As identified by Salah et al. (2019) [16], a sustainable and transparent supply chain needs a framework for tracking the origin, farming methods, and safety of the food product without resorting to a centralized third party.

Food Safety

Traceability allows for increasing food safety by assisting with traceable information in case of food contamination and by identifying untrustworthy actors.

Knowing all the transfers that each food product undergoes can contribute to preventing viruses or other contaminants [12]. In the case of a food recall, traceability information can be used to investigate and determine the probable cause, but also, as referenced by the European Union (EU, 2002) [14], to more accurately withdraw or inform affected consumers of contaminated products thereby avoiding a wider disruption.

Additionally, if on several occasions a supply chain actor is determined to be unreliable or their products have led to food safety issues, consumers should be able to access the company history before making the decision to buy their product, effectively tagging companies as trustworthy or fraudulent.

Efficiency

Again in the case of a food contamination scenario, efficiency is crucial. In a food supply chain, it is necessary to have fast detection and act quickly, but with the current structure of the supply chain, it takes up to 7 days on average to track the origin of a food contamination [15].

Moreover, the supply chain is saturated with intermediaries, and removing them could potentially reduce transaction costs and improve overall profits [10].

Food Origin

Customers are becoming more interested in knowing the origin of the food they are consuming. Regionally produced food and organic products are experiencing accelerated success amongst consumers, not only because these products are perceived as better and of higher quality, but because consumers want to contribute to sustainable methods of production and consumption [2].

Food Fraud and Counterfeiting

Food fraud is a significant concern since consumers' health is at stake [10]. Moreover, consumers are easily defrauded when it comes to food products as usually there is no visual property to distinguish these products. Regional regulatory bodies have implemented certification methods on specific local products. With over 400 certification systems for agricultural products in the EU alone, consumers may find it hard to differentiate and evaluate the credibility of each certification [15]. Additionally, these are usually easy to falsify and customers rarely go through the trouble of verifying existing certifications. The current model lacks a reliable and easy to verify, single certification method for agri-food products.

Food Quality

Agri-food products are usually highly perishable goods, any mistake by any participant in the supply chain can lead to final products of lesser quality. By monitoring the product throughout its lifecycle, and recording information such as storage conditions, businesses can better control the quality of their own products. [6]

Environment and Social Concerns

The increased environmental awareness greatly impacts current supply chains. Consumers want to move towards sustainable consumption but for an efficient transaction they need reliable information that allows them to identify the goods with less environmental impact, like production methods employed [12]. The implementation of traceability systems, particularly those emphasizing transparency, has the ability to holds all supply chain stakeholders accountable for their actions and negligence [17].

Additionally, solving the current information asymmetry issue, where participants are solely aware of their own transactions in the supply chain [12] will help consolidate initiatives such as Fairtrade with better tracking of associated participants and ensure they are attributed correct compensation [18].

The European Union (EU) took an important step aimed at strengthening the role of consumers wanting to achieve a more sustainable economy. Now the increased awareness and necessity for a

traceability system is encouraging the food industry to propose new technologies that are safer and more sustainable than the current ones [12].

2.2 Blockchain

Blockchain technology was first popularized by its use in Bitcoin [19], a decentralized digital currency system. Nakamoto proposed the use of this technology for its decentralization, making it the first currency system that does not rely on a central authority and still maintains essential characteristics like solving the double-spending problem.

The technology is typically described as a distributed ledger that is append-only, persistent, immutable, decentralized, and distributed.

A blockchain is composed of a sequence of blocks chained together with the use of cryptographic hash functions. Each block comprises multiple transactions grouped together, along with the hash of the previous block. This linking of blocks creates a chain, making it extremely challenging to modify an older block since any changes would require altering every subsequent block in the chain.

Appending new blocks is the main role of a special set of system nodes often called miners or validators. In the case of Bitcoin, when adding new blocks, miners must first solve a cryptographic puzzle. This task is designed to be highly CPU-intensive. The method is referred to as Proof-of-Work (PoW), and it prevents malicious nodes from altering previous blocks, as doing so would demand more CPU power than the cumulative power of all honest nodes combined [19].

Bitcoin, along with many others, is a permissionless blockchain system, meaning that the platform is public and everyone is allowed to participate without first interacting with an administrative authority.

In most permissionless blockchain systems, an account is primarily identified by its address, typically generated from a public key. This address, however, does not inherently disclose any details about the individual in charge of the account. The public key is derived from the account's private key, which is a unique key generated on demand and meant to remain undisclosed.

Additionally, access to the private key is the sole requirement for controlling an account. Each transaction submitted to the system is first digitally signed by the client using his private key, which by the properties of Public-key encryption causes the client to be the undeniable sender of the transaction.

To support the PoW consensus and its associated energy consumption, Nakamoto proposed an incentive system to keep miners interested in running the system. The system rewards the miner of each newly created block with Bitcoin's own currency.

Being public and decentralized means that no single organization owns the system and cannot modify it as they see fit, effectively establishing a trust-less system that relies solely on consensus algorithms to enforce cooperation between participants. It also provides better robustness as the trust model sup-

ports some misbehaving nodes without affecting completeness.

Ethereum [20] is another public and permissionless blockchain, meaning everyone can join in and become an active member of the system [21]. It also means that the information recorded is transparent and accessible to everyone.

Ethereum, like Bitcoin, has its own currency called Ether (ETH) but its focus is not the currency, instead, its philosophy is of a "world computer". It is an open-source, globally decentralized computing infrastructure that executes predefined programs called smart contracts. [22].

Smart contracts on the Ethereum platform are executed by the Ethereum Virtual Machine (EVM) and have their own high-level programming languages which compile directly into EVM-compatible bytecode. Solidity and Viper are examples of these programming languages. The EVM is considered Turing complete meaning that it can execute any program as long as it has sufficient resources to do so. It also means the EVM is susceptible to the halting problem, which states that we cannot predict whether a program terminates without running it. This is especially relevant in Ethereum because if smart contracts are allowed to run indefinitely it would disrupt the operation of the whole network.

One key aspect of Ethereum is the use of Gas to constrain the execution of smart contracts. Gas is a unit of measurement that represents the computational effort required to execute a particular operation on the Ethereum platform. Different operations require different amounts of Gas as specified by the EVM specification. The execution is limited by the amount of Gas provided which mitigates the halting problem by forcing the smart contract to terminate after all the Gas is consumed.

When a user wants to execute a smart contract on the Ethereum platform, they must specify a Gas price they are willing to pay in the transaction. This cost will be paid for using Ether. Higher Gas prices will result in transactions being added to the blockchain faster, as validators prioritize transactions with higher Gas prices.

Another relevant feature of Ethereum is the integrated events. Events, sometimes referred to as Logs, are a way of logging and transmitting data from within the smart contracts. This functionality enables users and external applications to monitor specific events and receive asynchronous notifications related to them. It is worth noting that these events are not accessible to the smart contracts themselves.

Smart contracts in Ethereum are created by a special transaction to the address `0x00...0`, containing the EVM compiled code, and are assigned a unique address. To execute a smart contract's function, a client sends a transaction to the smart contract's address specifying which function to execute and passing the arguments as transaction payload. The payload is usually the only input used by a smart contract. That is because the system requires a determinist execution of each function call so that other nodes' execution results in the same output state after they receive the transaction propagated by the consensus algorithm.

The use of Oracles can somewhat circumvent the previous challenge. An Oracle is a smart contract

that brings external data to the blockchain network, allowing other smart contracts to interact with the external world. They work by providing a consistent version of on-chain data for all nodes to access. This data is typically updated by a normal transaction, a state-altering function call to the Oracle contract, and the new value is disseminated to all nodes.

As of September 15, 2022, Ethereum has moved from a Proof-of-Work (PoW) consensus, similar to the one used by Bitcoin, to a Proof-of-Stake (PoS) consensus algorithm in order to improve sustainability and reduce the power consumption of the network "by around 99.95%"[2]. In PoS consensus[3], validators (previous miners) explicitly stake ETH tokens into a smart contract which are held as collateral if the validator acts maliciously. This major step in making Ethereum a more sustainable technology was named "The Merge".

Another challenge Ethereum faces is the high cost of interaction with the network. These costs have the potential to increase as Ethereum gains popularity and experiences higher network activity. This has led to the development of layer 2[4] solutions, which strive to reduce the cost and increase the scalability of the Ethereum network. A layer 2 is a separate, less congested, blockchain that extends Ethereum. One key feature of layer 2 solutions is the bundling of transactions into a "rollup bundle" before submitting them to the main Ethereum chain. This rollup approach effectively spreads the primary layer's transaction fees across multiple transactions, reducing costs for individual users.

In addition to layer 2 solutions, there are other offerings such as side chains[5] with corporations like Polygon[6] offering their Ethereum-compatible blockchain network that operates over a separate blockchain and offers cheaper fees and faster transaction times. The main difference between sidechains and Ethereum layer 2 solutions lies in their security models. Layer 2 solutions inherit their security from the main Ethereum network, whereas side chains rely on their independent security mechanisms and do not post state changes and transaction data back to Ethereum Mainnet. They are also EVM-based, meaning that developers are not required to make any changes to their code when experimenting with a side chain.

Hyperledger Fabric [23] is another blockchain technology that supports smart contracts code execution, denominated chaincode in this platform. It is open-source and maintained by the Linux Foundation[7]. Their goal is to provide enterprise-grade blockchain technology.

Different from the previous technologies, this one has a permissioned approach meaning that only authorized parties are allowed to access and participate in the network [24]. This quality makes it particularly suitable for business environments where privacy and security are often important concerns.

[2] https://www.ethereum.org/en/upgrades/merge/
[3] https://www.ethereum.org/en/developers/docs/consensus-mechanisms/pos/
[4] https://www.ethereum.org/en/layer-2/
[5] https://ethereum.org/en/developers/docs/scaling/sidechains/
[6] https://www.polygon.technology/
[7] https://www.hyperledger.org/

Hyperledger Fabric focuses on performance by moving away from the typical “order-execute” architecture to an “execute-order-validate” design. In contrast to Ethereum, which propagates transactions and executes smart contract’s code on every node in order to validate them, Fabric associates the chaincode with a specific set of channels, and only the peer nodes that are part of those channels are required to execute the chaincode. This allows for fine-grained control over which nodes are responsible for executing and have access to the full payload of a particular chaincode. It can help to improve the scalability and efficiency of the network, as well as its privacy.

Compared to Ethereum which has dedicated programming languages for writing smart contracts, Fabric allows for smart contracts to be written in a variety of common programming languages, such as Go, Java, and C/C++. This is made possible through the use of containerized environments in which chaincode is executed in a secure and isolated context.

To access stored data, the system offers a limited set of primitives, functioning like an Application Programming Interface (API). The only possible interactions the chaincode can have with on-chain data are through `GetState`, `PutState`, and `DelState` operations.

Following Ethereum’s introduction and its concept of smart contracts, numerous applications of blockchain technology have emerged, commonly referred to as Decentralized Applications (dApps)[8]. A defining feature of dApps is that their application logic, or ”backend,” is implemented through smart contracts. The primary advantages of dApps include transparent access to the backend code, safeguarding user privacy, and high system availability.

2.2.1 Decentralized Autonomous Organization

We end this chapter on blockchain technology with a final mention of Decentralized Autonomous Organization (DAO) [25]. A DAO is a type of organization that enables coordination between entities and operates on blockchain technology. It is decentralized in the sense that it is not controlled by any single individual or entity, but rather by a group of individuals who hold voting rights. Every decision in the organization goes through a democratic voting process that happens transparently on-chain, and so can be audited by anyone.

From decentralized investment funds[9] and decentralized exchanges[10] to governing physical land[11], DAOs can be used to manage organizations and their funds, while eliminating the need for trust between members. Instead, each participant relies solely on the transparency and immutability of the blockchain and on the democratized decision-making process.

[8] https://www.investopedia.com/terms/d/decentralized-applications-dapps.asp
[9] https://www.bitdao.io/
[10] https://uniswap.org/
[11] https://www.citydao.io/

A DAO's logic is encoded in a governance smart contract, typically running on the Ethereum platform, where smart contract code is public, auditable by everyone, and cannot be modified once deployed.

While it is a recent concept, different design models for DAO membership have been defined[12]:

- Token-based membership: Usually fully permissionless. To participate a member needs to hold a token that can be bought, traded, or earned.
- Share-based membership: Permissioned but open to applications. Individuals can propose to join and a defined share represents voting power.
- Reputation-based membership: Reputation defines voting power. Members cannot buy or trade reputation, it must be earned through participation.

Nevertheless, a DAO has its disadvantages. Security is a critical consideration. Unless a proxy is explicitly implemented, smart contracts cannot be periodically patched to address discovered vulnerabilities. Furthermore, since their code is public, it is also exposed to potential malicious actors. This makes DAOs especially vulnerable and, if sizable funds are involved, prone to attacks.

A famous case of an attack on a DAO was the "DAO hack" [26], where an undetected vulnerability in the smart contract code enabled an attacker to steal more than 3.6 million ETH, around 50 million dollars at the time, by repeatedly withdrawing funds from the DAO's treasury without member authorization. Additionally, DAOs can be slow and not the most efficient due to the need for several parties to vote on proposals prior to making a decision.

In summary, DAOs are a unique application of blockchain technology that enables decentralized coordination and decision-making.

2.3 Blockchain in Agri-food

There has been an accelerated growth in blockchain publications in the last few years. According to the Scopus database the number of published articles, referencing blockchain technology, has increased from 2 publications in 2010 to 7,005 in 2020 [12]. With each publication, researchers identify several advantages of applying this technology to different domains. Up until now it has been successfully applied in several industries, including, finance, healthcare, agriculture, manufacturing, and logistics, and has great potential to significantly change current supply chain management systems due to its decentralized architecture [9]. Whereas in the traditional supply chain, each member makes decisions independently and only interacts with adjacent actors, in a system supported by blockchain, data can be shared and synced giving participants access to information throughout the entire supply chain [5].

[12] https://ethereum.org/en/dao/

In fact, there is a rising trend in blockchain technology applied to the agri-food sector, with around 40 documents published in 2018 to almost 160 in 2020 [10]. With the assistance of several literature review papers [12, 10], we have selected a subset of the most relevant and feature distinct publications for analysis in this section.

In one of the first publications in the context of blockchain-based agri-food supply chain traceability, Tian [6] proposes a combination of RFID technology, for data acquisition, with blockchain technology to guarantee reliability and information authenticity. The main focus of the suggested traceability system was to strengthen the quality and safety of agri-food products in China's markets.

AgriBlockIoT was presented by Caro et al. [11] as "a fully decentralized traceability system for the Agri-Food supply chain management". Their system integrated the data acquired by various IoT sensors with the underlying blockchain technology. At that time, their proposal introduced the use of smart contracts in the context of the agri-food supply chain, a novel approach, as no prior publication had explored the autonomous transaction capabilities of blockchain technology.

Salah et al. [16] reference a need to create a secure framework for tracking details about the origin, farming methods, and safety of the food product without resorting to a centralized third party. Their work focuses on traceability and transparency as benefits for the consumer, and their goal is to allow a customer to access and analyze information related to the transversal of the product throughout the whole supply chain.

Another traceability scheme for tracking agri-food products from origin to the end consumer was designed by Shahid et al. [27]. Their work focuses particularly on developing a reputation system for monitoring the credibility of specific participants and providing a potential buyer with reviews that account for the seller's previous credibility.

For the Italian Carasau bread supply chain, Cocco et al. [17] propose a blockchain-based supply chain management system for region-specific products. They integrate the proposed decentralized application with custom-designed RFID technology to monitor each bread directly.

In the final publication we examined, Baralla et al. [28] specify a platform for guaranteeing the origin of region-specific food items with the use of blockchain technology in a Smart Tourism Region.

2.3.1 Use-cases

In this section, we will discuss what we consider the most relevant characteristics of the selected publications. Starting with a publication that defines a fundamental traceability framework and then exploring additional characteristics introduced in subsequent works.

Blockchain technology, with its decentralized attributes, could be the essential tool for creating a versatile agri-food traceability system that benefits both consumers and businesses. Indeed, significant players in the industry have taken an interest in this field. Walmart, an American multinational retail

corporation, collaborated with IBM to create a blockchain-based traceability system for pork in China and mangoes in the US [13]. Having had a positive experience, Walmart wanted to expand which led IBM to develop *IBM Food Trust*[14] project, a proprietary service that uses Hyperledger Fabric as the supporting blockchain technology. Amazon AWS has also developed a product for supply chain track and tracing[15]. Nonetheless, these proprietary frameworks are built upon the same foundational concepts as the publications discussed below.

AgriBlockIoT

Caro et al. [11] devices a system to give customers a complete history of the food they are buying (AgriBlockIoT) by following the *from-farm-to-fork* concept. Their architecture consists of three main modules:

- *API:* high-level layer of abstraction for integrating existing applications with AgriBlockIoT capabilities.
- *Controller:* translates high-level calls into corresponding low-level operations on the blockchain, such as querying and converting data stored on the blockchain.
- *Blockchain:* business logic implemented through smart contracts on the blockchain platform.

As mentioned previously, this work was designed to be blockchain independent and was implemented in two distinct blockchain technologies for performance comparison motives, which requires that the *blockchain* module be adapted according to the technology in use, while the others remain unaltered, fulfilling the modularity characteristic of the system.

They were the first to suggest the use of blockchain smart contracts along with IoT technology. Sensors acquire and submit data which in turn fire automatical smart-contract executions. The only pre-condition is that every new participant (entity or IoT sensor) is registered in the blockchain, this is, have their corresponding public/private key pairs to interact with the blockchain.

The system's performance was evaluated on three metrics: latency, CPU load, and network usage. They observed that Hyperledger Sawtooth performed better in comparison to the Ethereum counterpart. While Ethereum has the advantage of being more mature, it only supports a single transaction structure, whereas Hyperledger Sawtooth allows the definition of a custom structure, which could be the reason for the observed performance difference (around 30 times less network traffic and substantially less latency), as well as the fact that Hyperledger Sawtooth has fewer system nodes and implements a consensus algorithm which is more suitable for devices with lower computational power.

[13] www.hyperledger.org/learn/publications/walmart-case-study
[14] www.ibm.com/products/supply-chain-intelligence-suite/food-trust
[15] https://aws.amazon.com/blockchain/blockchain-for-supply-chain-track-and-trace/

Soybean Supply Chain

The next proposal discussed is tailored to the soybean supply chain. Salah et al. [16] defines the soybean supply chain as composed of seed companies, farmers, grain elevators (i.e., buys and stores grain), grain processors (i.e., refines grain), distributors, retailers, and customers.

Their solution focuses on traceability and monitoring of the soybean supply chain but in comparison to Caro et al. AgriBlockIoT (previous section [11]) they do not focus on providing the end customer with a full transparent history of the product. Instead, they design their system for continuous monitoring by the participating entities, triggering alerts in cases where a violation occurs. They argue that their framework is generic enough to be applied to almost all use cases of an agri-food supply chain involving multiple stakeholders.

Similarly to AgriBlockIoT, they rely on smart contracts deployed to an Ethereum public blockchain. To perform transactions in the supply chain each entity is assigned a unique Ethereum account address, which undeniably links the information to each submitting entity.

This system also focuses on submitting transactions between participants to smart contracts, which can later be used for tracking the movements of a product from participant to participant. But unlike AgriBlockIoT, the final customer is registered in the blockchain as the last entity in the supply chain, and their purchase is logged into the smart contract memory, finalizing the traceability chain of a product.

While both approaches we have seen record additional qualitative metadata (e.g., crop growth measurements), in the soybean framework they do not specify how this data is acquired (AgriBlockIoT references IoT sensors) and suggest a different approach of storing the data in a decentralized file system (IPFS) instead of directly in the blockchain. For example:

i) Farmers monitor the growth of their crop by regularly taking pictures, producing timestamped images, that are uploaded to the decentralized file system.

ii) The system returns the hash of the data submitted. Additionally, this hash is later used as a content identifier to access the data.

iii) Lastly the farmer submits the returned hash to the blockchain via a smart contract.

The data stored in the decentralized file system is verifiable since the accessing entity can compute the hash of the returned data and compare it to the hash stored in the blockchain. Uploading to the blockchain ensures the veracity of the files since a file uploaded by a farmer will make him/her the undisputed owner of the uploaded images and responsible for any inaccuracy or fraudulent images.

By using this method, the uploaded images can be audited and their content disputed by any participant in the supply chain. Using an external file system for storing large data is a technique that is also used in other works on agri-food supply chain [17] and is becoming common in blockchain applications

to mitigate the size constraints of a distributed ledger. We will explain the reasoning in more detail in Section 2.3.3.

Another characteristic of the solution is that it leverages events from Ethereum smart contracts as a notification system for interested parties. It specifies that the smart contracts are created by farmers. After an offline agreement between the farmer and the seed company, they execute the *buySeeds* function of the smart contract, specifying the farmer's account address, seed company's account address, and quantity. An event *SeedsRequestedByFarmer* is then triggered to inform all active participants of the farmer's intent to buy seeds, in which the seed company responds affirmatively by executing the smart-contract function *seelSeeds*.

The authors do not reference the reason behind having the smart contract created by the farmer. In our opinion, the framework should at least provide a template that has been previously safety tested. Nevertheless, there could be intentional changes introduced by a malicious farmer. In their solution, the negotiation phase has to be made offline before interacting with the blockchain. It could have been interesting to support the negotiation phase on the blockchain platform as a way to implement more enforceable contracts and maintain transparency with other participants in the supply chain.

In fact, they conclude their work by referencing that in the future they would like to integrate their proposed solution with automated payments and proof of delivery, where parties would be paid using crypto-currency on successful delivery.

Traditional Bakery (Carasau Bread)

Another case study is of the Italian Carasau bread supply chain [17]. This supply chain has some particular needs and characteristics that need to be constantly monitored and documented. Starting with the raw material used, the durum wheat, which cannot be tested for specific residues upon arrival at the milling industry, it needs to be accompanied by appropriate documentation from the supplier certifying that the necessary checks were carried out. Similar information needs to be documented for the flour, such as humidity, ash, and protein content. Information about storage conditions, such as temperature, also has to be logged for every responsible actor in the supply chain. Additionally, all documentation has to be kept and accessible by supervisory bodies at the time of inspection.

These requirements make the Carasau bread supply chain a perfect fit for new blockchain-based traceability systems. The authors propose a decentralized app (dApp) designed for the Ethereum blockchain, leveraging the use of smart contracts and RFID tags in conjunction with Raspberry Pi sensors for data acquisition.

Their proposed system is not only designed to support customer access to the traceability data but also as a system for the regulatory entity to audit the supply chain. The certifying authority for this use case is the Sardinian regional body. They are responsible for performing inspections, regulating

the supply chain, and authorizing access to the system, so only authorized actors can contribute with traceable data. Either batches or nodes can be the targets of an inspection. A node is approximately the same as an actor and on entering the supply chain they are assigned a unique identifier (Ethereum account address) and a parameter that indicates whether the node is compliant with the required quality.

On inspection, if the regulatory body finds that a node does not satisfy all requirements, they flag the node preventing other participants from dealing with a misbehaving one until necessary measures are enforced. Similarly to the Soybean use-case [16], their system leverages the concept of events in a smart contract to inform the affected nodes that an inspection has been performed and they need to act. Another similarity is that they also propose the use of an external file system, again IPFS, for storing documents associated with each batch and in doing so reducing on-chain data and associated costs.

Most of the available works are only theoretical proposals that authors plan on implementing in the future, not systems that have been used and tested extensively. Therefore these works might lack an explanation of critical aspects of the systems but nevertheless, they are useful to understand the common characteristics and the current state of the art.

2.3.2 Benefits

Several benefits regarding the application of blockchain technology to the agri-food sector were identified throughout the works previously discussed, such as monitoring of real-time data, supply chain transparency, increase in food safety, and minimizing health risks, among others [10].

Next, we will discuss the selected publications along with some additional works, in light of the agri-food requirements introduced in Section 2.1.2.

Traceability

Most works tackle traceability by recording product movements throughout the supply chain.

Tian [6] proposes collecting data throughout a product's lifecycle. Qualitative information like production region or pesticides used, as well as logistical data of operational staff is stored on RFID tags, from which the most relevant information is subsequentially uploaded into the blockchain to make it persistent.

AgriBlockIoT is a system that gives customers a complete history of the food they are buying by following the *from-farm-to-fork* concept. Caro et al. [11] proposed the use of IoT sensors to acquire and submit data, along with smart contracts to implement business logic and record information on the blockchain.

Salah et al. [16] suggests in their framework the use of timestamped images of the crop's growth as the metadata to be periodically uploaded which can be later used to validate the conditions agreed upon in deals with other participants.

Additionally, some publications [28, 17] reference the possibility of using GPS sensors to track a product in distribution.

Transparency

Transparency is a measure of how much information is shared between stakeholders. When a traceability system is configured to have its data publicly accessible, the supply chain is maintained fully auditable at any time, enabling the end user, the customer, to inspect detailed product information and its flow through the supply chain.

To access this information, Tian [6] suggests RFID readers to scan product tags at the supermarket. These would present the customer with details that could significantly enhance the consumers' trust and confidence in the product. Additionally, the author identifies potential benefits to other stakeholders of a transparent and open supply chain, for example, a logistics company could implement real-time tracking of products or a supervising regulator could examine the supply chain to determine the root cause of a defective product.

Caro et al. [11] suggests smart tags as a possible route for product identification, which in turn would enable customers at the time of purchase to transparently verify the history of a product throughout its lifecycle.

Cocco et al. [17] specify the goal of their proposed system to be transparent and auditable in such a way that each actor of the supply chain can verify that the product is in conformity with the hygienic and sanitary conditions.

Information Integrity

Data integrity, in the studied works, is supported by the use of blockchain systems that provide secure, immutable, and decentralized records.

According to Tian [6], the use of blockchain technology can provide reliability and authenticity to traceability systems by ensuring that the stored data is tamper-proof and verifiable. This is especially important in the context of a centralized information system, which may be prone to trust issues such as corruption or falsification. By decentralizing the supervision center and leveraging the decentralized and anti-tampering properties of blockchain technology, the credibility of the recorded information can be enhanced.

Salah et al. [16] also highlights the advantage of providing traceable functionality with unalterable information accessible to all stakeholders without relying on a central authority.

Food Quality

The related works that mention food quality describe the need for monitoring product conditions throughout the supply chain and possibly production methods.

Salah et al. [16] proposed a framework to guarantee the quality of the products that reach the end user by maintaining and monitoring the value chain between participants. They envisioned their smart contracts with the ability to check for violations and trigger events to all participants, for example, grains of different quality criteria cannot be mixed for sale since the total volume is known by all stakeholders.

Cocco et al. [17] system for Carasau bread supply chain management requires an authority to act as system admin. They suggested this role would be given to a regulatory body of regional products and it would have the power to inspect the product quality of individual batches or even participants, flagging them as unfit for the supply chain until corrective measures are taken. A similar proposal is made by Baralla et al. [28].

Food Safety

Similarly to food quality, authors propose that food safety could be increased by monitoring the supply chain conditions (e.g., storage temperature) and with faster recall times supported by a traceability system.

Tian [6] arguments that if safety and quality inspection centers, such as government departments or external third-party regulators, are also covered by the traceability system, along with other members of the supply chain, then these participants could also take advantage of the system by checking safety-related information at near real-time, allowing them to act quickly in case of a food safety accident and in doing so preventing further spread of the hazard. The authors propose that every participant is allowed access to the traceability information by being a node in the blockchain system that keeps an updated version of the blockchain, a reliable database.

Salah et al. [16] explains that, with the use of blockchain technology, the information submitted to the traceability system is permanently and irrefutably linked to its owner (sender of the transaction) and so every fraudulent or inaccurate data found can be disputed directly with the owner.

Cocco et al. [17] try to impose better food safety with their suggested inspections of the supply chain as explained in the previous “food quality” section.

Efficiency

Authors propose that new supply chain traceability systems could help make the supply chain more efficient and reduce transaction costs.

Blockchain technology and RFID tags could be used in conjunction to automatically add traceability

information to the system. This approach could decrease mistakes made by human factors as referenced by Tian [6].

Caro et al. [11] had the idea of leveraging edge devices such as gateways and IoT sensors and using them as nodes of the blockchain. They also designed their proposed system to be modular and blockchain-independent, while enabling integration with existing software (e.g. staff management software).

Finally, Baralla et al. [28] specifies, as a benefit to the transparency property of their system, that actors can share information more efficiently since everyone in the supply chain can track product-related information (e.g., ownership) all the time.

Food Origin

Most of the proposed solutions start collecting traceable information at the beginning of the supply chain since most of the time they want to provide the end customer with a full history of a product's traversal through the supply chain, including the origin of a food product.

Nonetheless, there are some implementations of traceability systems that are specifically designed with food origin in mind [28, 17], usually because they are region-specific products or are trying to mitigate counterfeiting of such products.

Additionally, Baralla et al. [28] states that information about a product's provenance exists but it is often hard to verify the trustworthiness of this information so they propose a blockchain-based system to solve that.

Food Fraud and Counterfeiting

Blockchain technology has been identified by some authors as a tool for mitigating counterfeiting in an agri-food supply chain. We believe that the implementation of such a system could provide a more secure certification system for agri-food products.

F. Tian (2016) [6] shortly suggested encrypting product data and binding the product with a unique ID RFID tag to prevent counterfeiting, noting that the problem resides in securely identifying the product and not in the risk of data manipulation since blockchain technology prevents it.

Salah et al. [16] explains that in their system it is possible for a stakeholder to record fraudulent data, but in the blockchain, it is always possible to link the data to its owner with 100% certainty, and so at a later stage when this data is found incorrect other participants can take action (penalties) on fraudulent stakeholders.

Shahid et al. [27] states that blockchain technology has a trust-less nature and that entities may act maliciously. To overcome this challenge, they focus on developing a reputation system for the agri-food supply chain, where buyers can assert the credibility of a seller by accessing previous reviews.

Whenever a participant transacts with a seller, he/she decides on a ranking and submits a review of the product owner which is later made available to all other participants as a trust value. They conclude by identifying that this system is not fail-proof against fake or biased reviews so they plan to integrate a fake review detection system within their reputation system.

Baralla et al. [28] designed a blockchain-based system for certifying local agri-food products to prevent counterfeiting and promote the regional economy. Traceable information about a product is stored on the blockchain and for each product, customers can verify this information by scanning a QR code on a specialized mobile app.

Environment and Social Concerns

Although not extensively mentioned by the reviewed works, the use of blockchain-based traceability systems can, with access to data from each stage of the supply chain, contribute to "more accurate shelf life of food products leading to reduced economic loss and food waste" [29].

2.3.3 Challenges

During system design, authors mostly identify similar challenges of applying blockchain technology to supply chain management. In this section, we discuss the challenges we consider most relevant.

The first challenge, and common to other applications of blockchain technology, is that the cost of running a blockchain-based traceability system could result in a price increase of the covered products [8]. However, as depicted in Section 2.1 customers are willing to pay a premium price for reliable traceability data of products. Nevertheless, the cost is great and is increasing for platforms like Ethereum due to its recent boost in popularity. A smart-contract creation alone will cost around 1.22€ plus 0.006€ per byte of code, assuming Gas price of 15 Gwei and 1,540€ per Ether. Additionally, it costs 20,000 Gas to store 256 bits in smart-contract storage, which is around 1.85€ per kilobyte.

Besides storage costs, it is necessary to consider the transaction cost when using public blockchains like Ethereum. Some works [28] provide a price breakdown of the described solution but this was calculated with an ETH price that is about 10 times less than the current price, this means that at times a single supply chain transaction in their system would only cost cents but now each transaction would cost a couple of euros in Ethereum Mainnet.

To circumvent this challenge, the use of external file systems has become popular in proposed works, with most using IPFS[16] for a decentralized platform that offers storage at a lower cost. The idea is to upload documents produced along the supply chain (e.g., images, videos, PDF) directly to IPFS which responds with an identifier to access the uploaded content. This identifier additionally contains the hash

[16] `https://ipfs.tech/`

of the uploaded content and by storing this hash on the blockchain, it not only can be used by an auditing user to access the content but also to verify the integrity and ownership of the uploaded documents.

Scalability, lack of regulation, and lack of training are also identified as challenges for the employment of this technology in the agri-food sector [10]. However, despite these challenges, the use of blockchain technology has the potential to enhance traceability systems and improve the credibility of recorded information. Additionally, as blockchain technology becomes more mature and is developed toward sustainability, it becomes a more viable option for use in supply chain management.

2.3.4 Opportunities

As new technologies are adopted in the agriculture industry, new opportunities emerge. Agriculture 4.0 has been defined as the next evolution of the industry, where technology like Artificial Intelligence (AI) and IoT will help increase productivity, prevent food waste, help manage resources, and adapt to climate change [30]. A blockchain-based traceability system can provide enormous amounts of data such as weather, soil, and storage conditions as well as marketing demands, which can be later used for data mining or Machine Learning purposes.

As for data mining, anomaly detection and fraud detection were identified as the most productive applications [31] but we imagine that they could also be effective in determining good farming methods based on the data submitted by farmers.

Machine Learning, specifically Deep Reinforcement Learning in conjunction with a blockchain-based traceability system was considered to make effective decisions on production and storage for profit optimization [32].

2.3.5 Summary

While most of the reviewed works focus on traceability, transparency, and information integrity, over the years authors have started to expand the reach of their solutions to other relevant consumer requirements, as demonstrated by Table 2.1 which presents a summarized view of the requirements supported by each work.

Proposal	Trace.	Trans.	Integ.	Qual.	Safe.	Effic.	Orig.	Fraud	Envir.	BC type
Tian [6]	✓	✓	✓					✓		Public
Caro et al. [11]	✓	✓	✓			✓				Both
Salah et al. [16]	✓	✓	✓	✓	✓		✓			Public
Shahid et al. [27]	✓	✓	✓	✓			✓	✓		Public
Cocco et al. [17]	✓	✓	✓	✓	✓		✓			Public
Baralla et al. [28]	✓	✓	✓		✓	✓	✓	✓		Public
Tracer (our solution)	✓	✓	✓	✓	✓	✓	✓	✓	✓	Public

Table 2.1: Summary of requirements addressed by each article plus our proposed solution.

The metric for populating the table includes any explicit mention of a customer requirement or any requirement that can be partially aligned with the given definition. Each of the table columns refers to a customer requirement defined in Section 2.1.2: traceability, transparency, information integrity, food safety, efficiency, food origin, food fraud and counterfeiting, food quality, and environment and social concerns. The last column depicts which blockchain type is used in the proposal.

The table is organized chronologically by proposal date, and there is a clear distinction that older proposals concentrated on fundamental requirements, while newer ones built upon the foundation by introducing additional features.

Our investigation revealed that no existing system comprehensively fulfills all the requirements gathered during our analysis. In contrast, our proposed system places a strong emphasis on meeting customer needs and aims to effectively address all the identified requirements.

3

Tracer: Design and Implementation

Contents

Tracer[1] is a system that leverages Ethereum-based smart contracts to establish a traceability framework, systematically recording the journey of goods throughout the supply chain. Moreover, this system incorporates a DAO to supervise the operations within these supply chains. Tracer is designed to offer adaptability across a spectrum of supply chains, with a primary focus on the agri-food sector, which is the central theme of this dissertation.

In the upcoming sections, we will review the design of our system components and implementation decisions.

3.1 Design

One main characteristic of the Tracer system is the use of blockchain technology, more specifically, smart contracts executing on the Ethereum platform. This technology allows for the creation of a transparent

[1] https://github.com/DuBento/Tracer

and immutable record of transactions, providing a secure and reliable mean for tracing product movements throughout a supply chain. The decentralized nature of blockchain technology ensures that the information recorded cannot be altered or tampered with, providing a high level of trust and confidence in the traceability system.

Furthermore, the transparency associated with a public blockchain can help increase the efficiency of our traceability system as the data can be accessed at any time, enabling faster product identification in the case of a food hazard outbreak.

Additionally, the use of blockchain technology can help to increase the efficiency of our traceability system, as data transparency translates into being constantly accessible by all parties involved, enabling faster food recall in case of a food hazard outbreak.

Overall, our traceability system can help to improve the transparency and accountability of supply chain actors, allowing consumers to make informed choices about the products they buy. We attempted to create a system that can be highly trustable, leveraging the security and decentralization properties inherited from the blockchain technology.

Our solution distinguishes itself from other works in the field by proposing the use of a DAO as the regulatory body of our traceability system. The inclusion of a DAO empowers Tracer with decision-making and governance capabilities, thereby equipping it to potentially manage entire supply chains in a decentralized manner. Furthermore, this organization can be employed to enforce supply chain standards as it has the authority to revoke access to misbehaving actors, thereby promoting food quality and combating fraud.

3.1.1 Use Case

To better explain the use of our system, from a client perspective, we specify one use case supported by our solution.

We can consider a use case where a consumer wants to purchase a wine bottle from a wine shop. The wine is produced by a specific winemaker in a certain region. The client can access our system's interface by scanning the QR code present on the wine bottle. This action will redirect the user to a page displaying detailed information about the supply chain journey of the product. They can see the product's traversal between entities in a from-farm-to-fork scenario, in this case, from the vineyard to the wine shop.

The customer can check the origin of the wine and ensure that they are indeed produced from the region claimed. They are also assured that the product and producer are correctly certified by the regional regulatory body, as the producer requires previous authorization from the regulatory body to interact with the traceability system it oversees. Additionally, any transgression flagged by the regulatory body would be visible to everyone, including the user, due to the transparency of our system.

In the same interface, consumers can check additional information recorded as partial updates during the product lifecycle. These updates can include anything the actor desires. They could, for example, provide incremental updates about the wine aging stage, including information such as liquid density, temperature, and even images depicting the wine's color. Every update is accompanied by a trustable timestamp, providing the client with a reliable time interval for a specific stage.

Similarly to the updates from the traceability system, the client can also access additional information about actors involved in the supply chain, such as yearly CO_2 emissions. This allows customers to make a more informed decision, ensuring that they are supporting businesses that adhere to social and environmental norms.

Additionally, the customers want to know if the batch for this wine has been flagged for any potential hazard, such as product contamination. And, if an outbreak is identified, retailers can quickly proceed to recall the wine. Often, news of outbreaks is disseminated through various communication channels, yet we rarely receive direct notifications regarding products we own. By implementing the Tracer system, we empower consumers to verify the status of a product, even after purchase, and determine whether it belongs to an affected batch.

Multiple initiatives, including companies like Everledger[2] and Compellio[3], which center their business around a similar use case, emphasize the realistic nature of this use case and the tangible value it offers to consumers.

Overall, it highlights the various properties that we want to provide with the traceability system, including transparency, food quality, origin and safety, efficiency, and the ability for consumers to make informed decisions about their purchases avoiding fraud and counterfeiting and supporting businesses that adhere to social and environmental norms.

3.1.2 System Architecture

The diagram in Figure 3.1 illustrates the interactions among the various components that constitute Tracer.

In the diagram, we distinguish between three types of agents:

- The "Customer', represents the client who has acquired or is considering acquiring the product.
- The "Supply Chain Actor", encompasses intermediate stakeholders who handle the product throughout its lifecycle, such as producers or distributors.
- And the "DAO Member", refers to users that are recognized as members and have the ability to propose and vote on actions within the DAO. More details about DAO members are provided in

[2] https://everledger.io/
[3] https://compell.io/

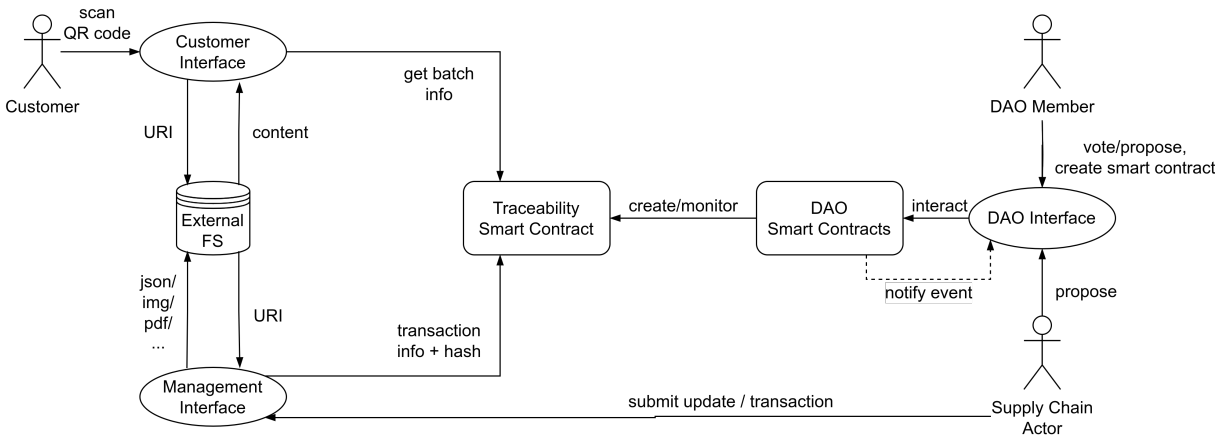


Figure 3.1: The interactions between users and system components

Section 3.1.3.

The system comprises several smart contracts running on the underlying blockchain technology:

- The “Traceability Smart Contract”. This contract is responsible for recording essential information to enable product traceability. This particular smart contract is detailed in Section 3.1.4;
- And the “DAO Smart Contracts”. These are a collection of contracts that serve to support the functioning of our DAO. It consists of the following individual components: a “Governor Contract”, in charge of managing proposals and votes within the DAO; an “Executor Contract”, to execute successful proposals; an “User Registry”, designed to manage information related to members and other stakeholders; And a “Traceability Contract Factory”, used to create new instances of the traceability smart contract.

Tracer also integrates an external file system (“External FS”) for storing additional large data chunks, minimizing on-chain data storage.

Finally, our system provides web-based user interfaces that facilitate interactions with the underlying smart contracts.

- The “Customer Interface” offers a graphical representation of the product’s journey through the supply chain and the involved stakeholders.
- The “Management Interface” allows supply chain participants to manage product data and directly interact with the traceability system.
- The “DAO Interface” facilitates the governance process by enabling proposal submission and voting.

3.1.3 Decentralized Autonomous Organization: Regulatory Body

Decentralized Autonomous Organizations (DAOs) can be used to facilitate various types of organizations, including cooperatives and non-profits, leveraging the fact that they are self-governed and do not rely on a single entity to make decisions.

We propose the use of a DAO as the regulatory body of the supply chain traceability system. We envision that the members of our DAO could be existing food regulatory and inspection agencies that are already responsible for supervising the supply chain. These organizations, which governments typically establish at the national or regional level, could include agencies such as the U.S. Department of Agriculture (USDA)[4], The Food and Drug Administration (FDA)[5], The European Food Safety Authority (EFSA)[6], among others.

By using a DAO to oversee the regulatory process, these agencies can maintain their independence when it comes to proposing actions, but for a decision to be made, the proposal has to go through a collective voting process involving multiple members.

The decentralized nature of the DAO also allows for greater transparency and accountability, as all votes support non-repudiation due to blockchain properties. Additionally, our DAO can define a reputation for each member within the DAO that could be taken into a weighted vote scenario, with agencies with a broader scope (several markets) being given a higher reputation and more influence in the process. National bodies with the second-highest reputation and regional bodies with the third-highest reputation. It is important to note that this is only an example, and the reputation of members within the DAO could be configured differently depending on the specific goals.

In addition, the DAO is designed to allow outside agencies to join, by having existing members deliberate on signup requests and determine whether to admit the new agency as a member.

In Tracer, each regional body will manage an instance of a traceability smart contract (described in detail in Section 3.1.4) to track and record transactions within the supply chain they manage.

By analyzing the recorded traceability information or through periodic inspections conducted by these established regulatory agencies, they can, as members of the DAO, identify problematic stakeholders or batches and suggest actions to be taken. Once a decision is made, our system flags all the affected products in real-time. As a result, the next customers accessing the traceability data will be alerted to the problem, facilitating a quicker response to food safety outbreaks.

The conformity state of an actor, or of a batch, can be one of the following: `Functioning` or `Corrective measure needed`. If any wrongdoing is detected by the DAO, the supply chain actor can be flagged and their activity suspended. The affected participant is then notified that corrective measures need to be taken and their conformity state is changed to `Corrective measure needed`. The participant can then

[4] https://www.usda.gov/
[5] https://www.fda.gov/
[6] https://www.efsa.europa.eu/

notify the DAO that measures have been taken or refute the actions, by proposing that its state be reinstated and providing an explanation. If the DAO approves the changes, the participant's conformity state is returned to `Functioning` and they are allowed to resume their activities. All of the DAO's decisions are made as a collective and require votes from several bodies, ensuring that no single agency can block the activity of a particular supply chain participant.

However, it is important to note that the regulatory body may not continuously inspect the state of the supply chain and that the proposed system does not fully mitigate the insertion of fake data by supply chain participants. Nevertheless, for this data to have any meaningful effect it needs to be inserted into an ongoing supply chain, which means that the receiving participant might be able to detect inconsistencies in the received goods. To address this issue, the system admits individual supply chain actors as proposers, meaning that when inconsistencies are detected they can submit a request for the DAO to investigate.

The DAO's logic is encoded in a smart contract with predefined functions, attributes, and events. The diagram in Figure 3.2 represents the relevant parts of the DAO. As previously mentioned, the DAO consists of multiple individual smart contracts, but for the sake of clarity, we have represented a generalized version of the DAO within a single contract.

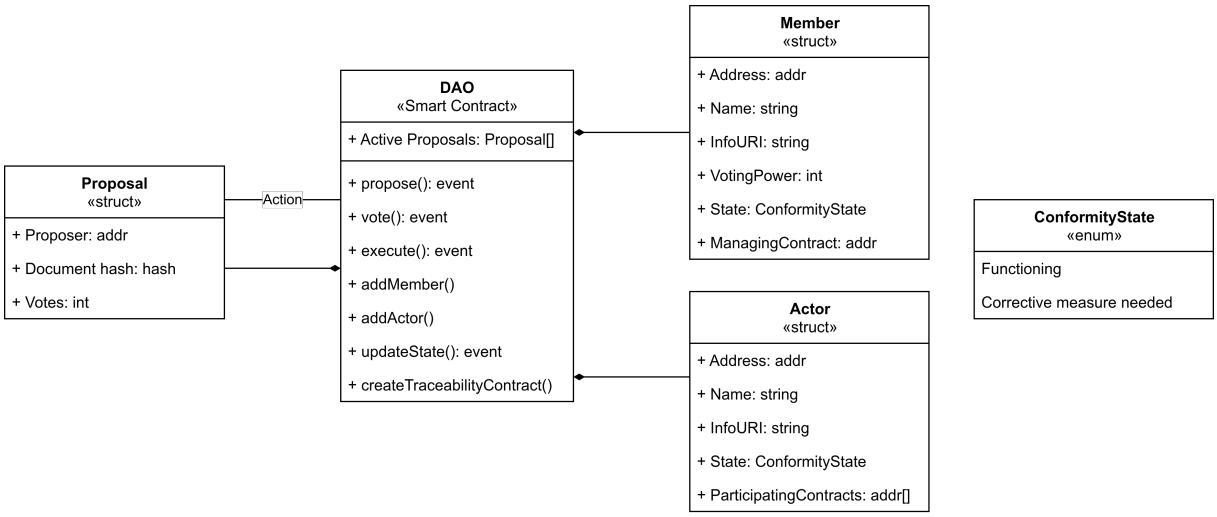


Figure 3.2: Class diagram for proposed DAO with functions and attributes specified

It is worth noting that the DAO records members' and supply chain actors' basic information but can also store additional information about these entities in the Uniform Resource Identifier (URI) that links to our external storage system. Any relevant information like CO_2 emissions and relevant certificates, could be linked to a member of our system and provide consumers with a common single source for information of this type and could support more informed decision-making based on existing environment or social concerns.

Overall, the use of a DAO as the regulatory body of our traceability system offers a promising solution as it enables a decentralized approach that maintains the benefits and maturity behind some existing organizations and provides a transparent and accountable means for regulating the system. Note that if the use of the Tracer system is popularized, the DAO has the potential to function as a democratized entity to oversee the entire supply chain.

3.1.4 Traceability Smart Contract

The traceability smart contract is a key component of the proposed system. It records information about the product lifecycle to facilitate subsequent product traceability and enforces the defined constraints.

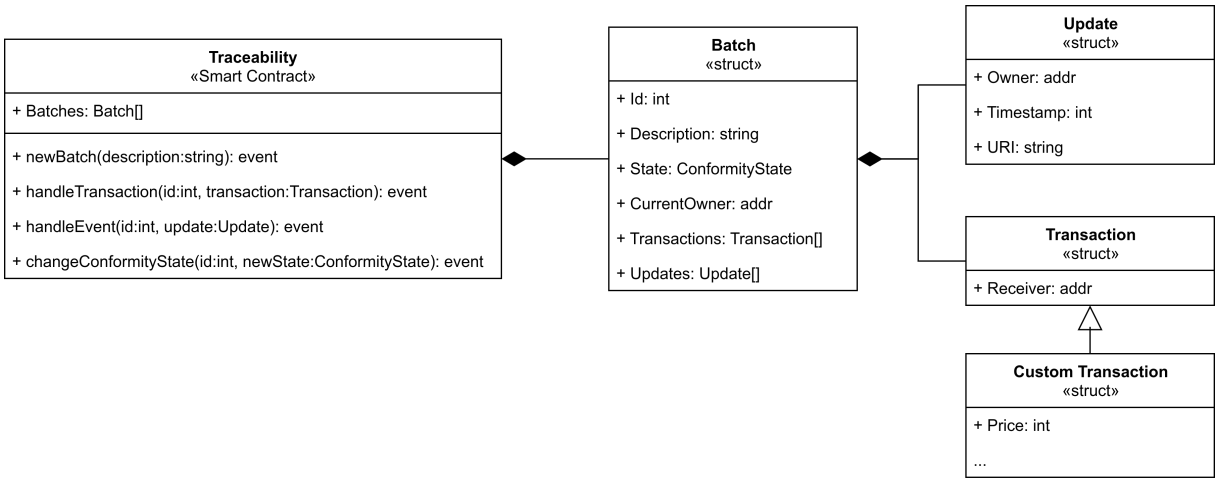


Figure 3.3: Class diagram for the proposed traceability smart contract

The diagram in Figure 3.3 represents the main components of our traceability smart contract. At the center of the diagram is the Batch structure, which represents a group of products that are being tracked within the supply chain. A batch generalizes the data for the whole batch of products, reducing the amount of data that needs to be stored, while also allowing individual products (such as handmade or luxury items) to be tracked separately by creating a batch for an individual product. Each product batch is identified by a QR code containing the smart contract address and batch ID.

In addition to batch-specific data, the batch structure includes “Updates” and “Transactions”. The `handleUpdate` method facilitates the submission of incremental updates, which serve to enrich the traceability system with additional non-critical information such as crop growth, storage conditions, etc. Each update includes a combination of on-chain and external information, as will be explained in Section 3.1.5. The information in these updates approximates the consumer to the production process and empowers them to make better-informed purchase decisions.

On the other end of the spectrum, a transaction represents an exchange between two actors in the supply chain. The transaction structure includes essential logistic details for our traceability system,

like the receiving actor and timestamps. To record a transaction, the `handleTransaction` method is employed, which can also generate a new update containing additional details regarding that transaction.

Our system provides a general framework that can be applied to most types of supply chains. This framework includes a template for generic transactions, featuring the essential data as previously outlined. Additionally, it can be easily expanded to include specific data as required by managing members.

Tracer supports the creation of custom transactions, which are essentially extensions of the standard transactions, tailored to meet the requirements of specific regulatory bodies. For instance, in Figure 3.3, we illustrate a CustomTransaction structure that includes data related to the price paid for a product in that particular transaction. Such details may be required by organizations like Fairtrade, which seek to ensure fair and sustainable outcomes for all participants within the supply chain.

Each operation of our system incorporates constraint checking to ensure the integrity and security of the data. Aside from checking if the actors are allowed within that particular smart contract, we also maintain a record of the batch's current owner, signifying the individual or entity currently holding the product. Only the current owner is authorized to append updates or transactions within the system.

This ownership constraint ensures that product-related actions are executed solely by the legitimate holder of the item. Only when a transaction transfers ownership to a different party can that new owner perform subsequent updates or transactions.

The traceability contract is intentionally defined in an abstract quality, providing only essential and basic operations, to maintain the ability to be used in different supply chains.

3.1.5 External File System

There are several reasons to use an external file system in conjunction with the blockchain platform. The use of an external file system in our traceability system allows for the storage of additional information at a reduced cost, compared to on-chain storage. A detailed comparison will be carried out in Section 4.3.1 of our evaluation.

This external storage can be in the form of a decentralized platform like InterPlanetary File System (IPFS) or cloud storage services like Google Drive, Amazon Web Services (AWS), and Dropbox which provide scalable, reliable, and secure storage options.

One important aspect to consider when using an external file system is the security of the stored data. Hashing algorithms, like SHA-256, can be used to secure the data by ensuring its integrity, by producing a unique hash for that specific content. Any changes to the content will result in a completely different hash value. The resulting hash is stored in the blockchain system and will inherit the properties of the underlying platform, such as the integrity of the hash since after submitted it cannot be modified by another actor. Most importantly, access to the hash allows viewing actors to verify the integrity of the data and be certain it was submitted at the recorded time.

The external file system can be used to store information like crop growth details or storage conditions for products, as well as other documents produced by the supply chain (e.g. receipts). Overall, the external file system allows for additional information to be added to the traceability system, possibly ensuring better quality and safety of the products in the supply chain.

3.1.6 User Interface

In Tracer there is a distinction between three web-based interfaces, each with its underlying focus.

An interface for actors to submit transaction events and product updates (Management Interface in Figure 3.1):

- This interface is used by businesses to interact with our traceability smart contracts.
- IoT devices could potentially be used for this purpose, but for the purpose of this work, we have chosen to use a web interface for manual data submission. However, an API is provided to enable integration with IoT devices or other existing supply chain applications if desired.

Figure 3.4 represents the final iteration of our management interface.

A second interface for customers to visualize the traceability data (Consumer Interface in Figure 3.1):

- This interface is used by clients to view the full product's history, including actors involved, and transaction times.
- The client accessed the interface by scanning a QR code that leads to a web page. We think this interaction might result in a higher adoption rate compared to the use of a specialized app since consumers are getting used to scanning QR codes for information.
- The user can choose to expand a specific transaction for more details of that stage. I.e. display submitted updates.
- On request, the user can view additional information about the member and supply chain.
- From the analyzed options [33] we chose to support a timeline design as we reason that a time-ordered list could facilitate data visualization.

Figure 3.5 displays the implemented consumer interface, featuring a timeline design.

A last interface for governance where DAO members vote and propose actions (DAO Interface in Figure 3.1):

- This interface is used by members of the DAO to submit proposals and vote on them.
- It displays information about the proposal, including the entity (or entities) involved and the reasons for the submission (request for action).

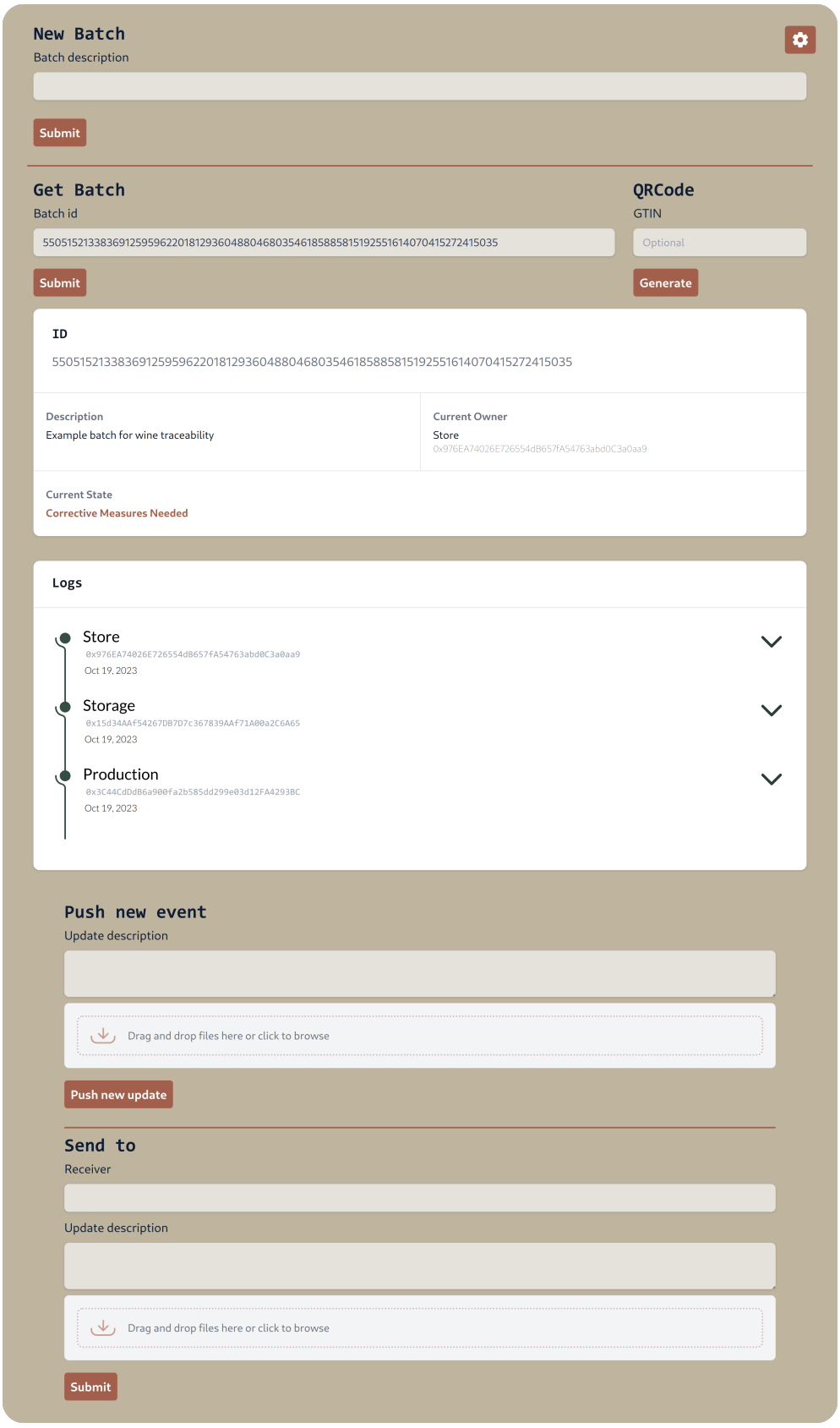
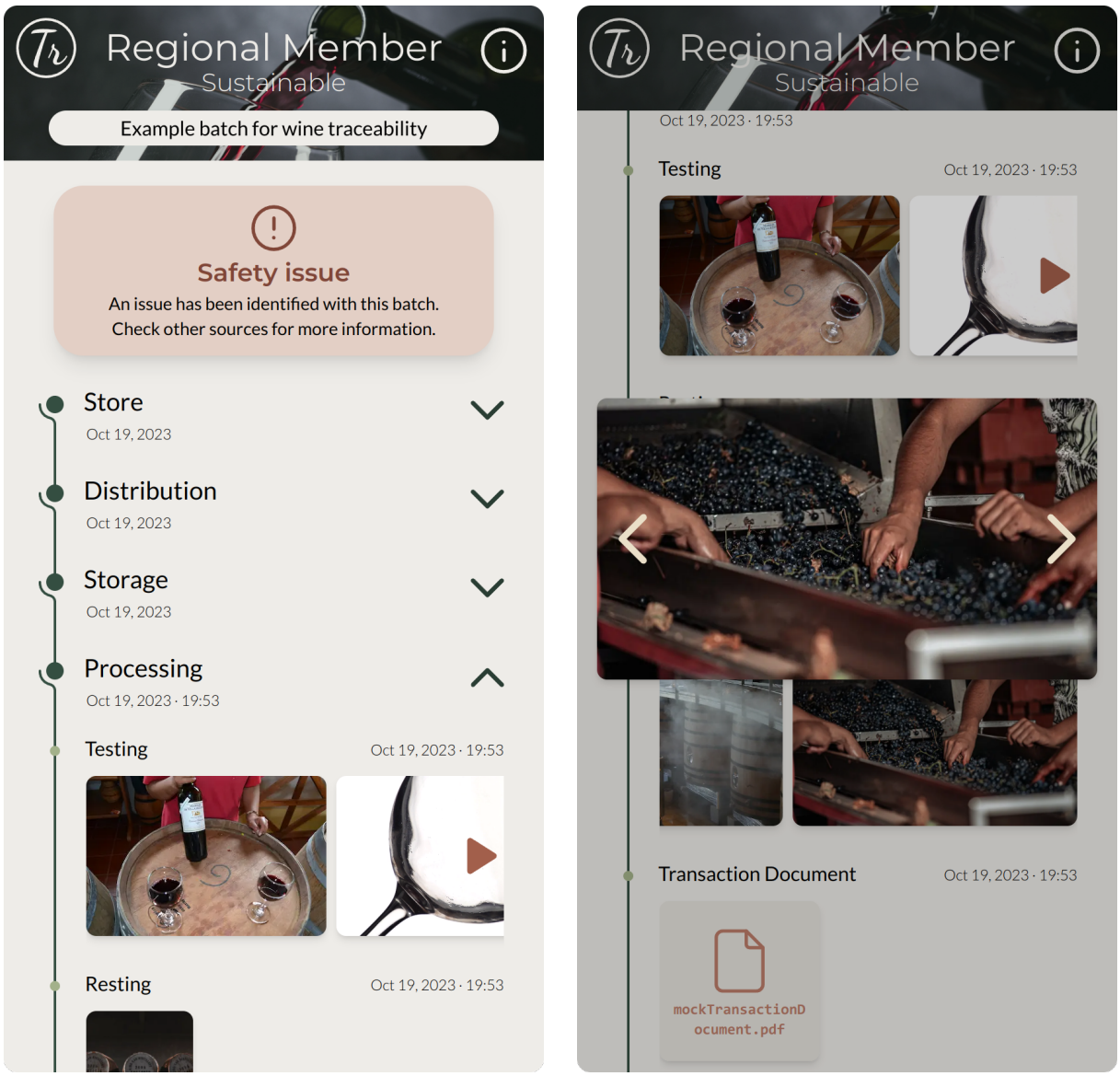


Figure 3.4: Management interface



(a) Timeline design

(b) Expanded image

Figure 3.5: Consumer interface: mobile device view

- Provides a dedicated page for active proposals sorted by time with voting operation available.
- Another page for adding new members to the DAO, with the possibility of uploading additional member-related data.
- And a final one for the traceability contract creation, which allows members to choose additional attributes to be enforced on their traceability contract instance's transactions.

Regrettably, because of time constraints, we were unable to complete the implementation of this last

interface.

3.2 Implementation

To implement the Tracer system, we utilized different technologies. For blockchain development, we used the Hardhat[7] development environment with the following extensions: Hardhat-ethers – for Ethers.js integration; Hardhat-deploy – for smart contract deployment; Gas-reporter – for reports on Gas usage (transactions and deployment); Typechain – for generating Typescript binding for our smart contracts;

The smart contracts were written in the Solidity programming language and deployment scripts and tests were written in Typescript. The developed system was intended for the Ethereum network but it can be used and deployed to any EVM compatible blockchain.

As for the web interfaces, they were developed using the same technologies, in fact, they were developed into a unified web server. These technologies are: Next.js[8] – an open-source React framework with a particular focus on server-side rendering; React – a library for web user interfaces; Tailwind-CSS – a CSS framework for style web interfaces; Ethers.js – a library for interacting with the Ethereum blockchain;

All components, routes and API were developed in Typescript.

In the following sections, we will analyze some of the implementation decisions made.

3.2.1 Traceability Smart Contract

This section focuses on the implementation choices of the traceability smart contract.

Batch Identifier

As explained before, the data unit of our traceability system is a batch. This unit represents a collection of products that are processed together in that supply chain. In our contracts, we will be storing several instances of a structure that represents a batch in a Solidity type called “mapping”. For the reasons that support this choice, refer to Section 4.5.

To access data in a `mapping` we need the corresponding key. The following snippet is used to generate a key which we will use to identify each batch.

```
uint256(keccak256(
    abi.encode(payloadHash_, msg.sender, block.timestamp, block.prevrandao)
));
```

[7] https://hardhat.org/
[8] https://nextjs.org/

The above logic is used to create a pseudo-random number. As referenced in Section 2.2, in the Ethereum network a transaction has to produce the same result in all validators to satisfy the consensus protocol and keep a consistent state. This means that a method call has to be deterministic, and so a random number generator cannot be fully random, it has to generate the same number on all nodes. To address this we devised a generator that utilizes block and transaction data:

- `payloadHash_`: represents the keccak256 hash of the `newBatch` method arguments (batch description and document URI).
- `msg.sender`: represents the address of the entity that sent the current transaction.
- `block.timestamp`: indicates the timestamp in Unix time from when the current block was added.
- `block.prevrandao`: is an opcode introduced in EIP-4399 (after The Merge) that represents the random number used in the Proof-of-Stake protocol of the previous block.
- `abi.encode`: a function that encodes its arguments in a byte array according to the ABI encoding rules.
- `keccak256`: is a cryptographic hashing function that generates a fixed-size 256-bit hash value, similar to the more commonly known SHA-256.
- `uint256`: finally the 256-bit hash is typecasted into an unsigned 256-bit integer to be used as a number.

This approach will effectively create a pseudo-random number that will produce a different value for distinct transactions.

EIP-6372: Contract Clock

It is also relevant to point out that, our system incorporates timestamps for both updates and transactions, a critical feature essential for ensuring accurate data presentation in the chronological format of the client-view interface. Many contracts rely on some form of clock to manage historical data or enforce delays. There is no standardized approach to achieve this, some contracts rely on block numbers, while others use block timestamps. Unfortunately, there is no programmatic means to determine which time-tracking method a contract employs.

To address this issue, Ethereum Improvement Proposals (EIP)-6372 has been propose[9]. It's worth noting that this EIP is relatively recent and currently under review. However, for the sake of future-proofing our contract, we have chosen to support it.

Contracts compliant with EIP-6372 must implement two specific methods:

[9] `https://eips.ethereum.org/EIPS/eip-6372`

```
function clock() external view returns (uint48);
function CLOCK_MODE() external view returns (string);
```

The `clock` returns “uint48” which is adequate for storing realistic timestamp values. Specifically, in timestamp mode, “uint48” remains sufficient until the year 8921556. Opting for a data type smaller than “uint256” facilitates storage packing of timepoints along with other associated values. This optimization can reduce the cost of reading and writing data to storage, a topic we will discuss further in Section 4.5. Using a type smaller than uint256 allows storage packing of timepoints with other associated values, greatly reducing the cost of writing and reading from storage, further discussed in Section 4.5. The `CLOCK_MODE` function, on the other hand, returns a string that defines the internal method used. Since we decided to use block timestamps, it must return ”mode=timestamp”, informing external contracts of the internal method used.

EIP-1167: Minimal Proxy Contract

We have implemented a Clone Factory as part of our system to efficiently create new instances of traceability smart contracts. This Clone Factory, also known as Minimal Proxy Contract, was proposed as part of the EIP-1167[10]. This standard defines a minimal bytecode implementation that redirects all function calls to a specified, fixed address of a default implementation of the Traceability contract.

To better understand how this works, we have first to comprehend the “delegatecall” operation. In Ethereum, there are two main types of calls: “call” and “delegatecall”. A “call” executes a function in another contract while keeping its own state separate. In contrast, a “delegatecall” also executes a method from another contract but it is executed in the context of the calling contract. In essence, it dynamically loads and executes code from a separate address at runtime. Storage, current address, and balance still refer to the calling contract, only the code is taken from the called address. Meaning that when a function is invoked using a ”delegatecall,” it can access and modify the state of the calling contract, transaction sender (i.e. ”msg.sender”), and ETH tokens associated with the transaction (i.e. ”msg.value”).

In fact, when using our Clone Factory, there is only one version of the traceability smart contract bytecode deployed on-chain and all other instances from our factory delegate call to this default implementation. This implementation forwards all calls and 100% of the Gas to the implementation contract and then relays the return value back to the caller.

This approach significantly reduces deployment costs by allowing multiple contracts to share the same bytecode, making them more efficient and cost-effective. In fact, the total bytecode size of each instance created from our factory is 45 bytes, more specifically:

[10] `https://eips.ethereum.org/EIPS/eip-1167`

```
363d3d373d3d3d363d73[<address>]5af43d82803e903d91602b57fd5bf3
```

Where the bytes at indices 10 - 29 (inclusive) are replaced with the 20-byte address of the default traceability contract. In the upcoming Section 4.3 of our evaluation, we will discuss the improvements in Gas consumption achieved by implementing the clone factory, mainly a noteworthy reduction of 86.36% Gas used when creating a new traceability smart contract.

Additionally, as stated in our system's design, we want to give traceability contract managers the ability to require additional information to be included within each transaction. Unlike typical frameworks, where users statically define structured parameters prior to deployment, our approach employs a Clone Factory, meaning that our transaction template (i.e. in the default implementation) cannot be statically changed as its code is pre-deployed and "inherited" by all instances.

To circumvent this limitation, our solution offers a dynamic parameter that can be configured during contract initialization. The default implementation provides a transaction structure, that by default only includes an address for specifying the transaction receiver, and a dynamic array for any supplementary mandatory attributes. This dynamic approach ensures that our traceability system remains versatile and can be easily tailored to meet specific requirements while still benefiting from the cost-efficient deployment facilitated by the Clone Factory.

Update Events

In the initial implementation phase, we designed the traceability smart contract to store data URIs directly on contract storage for each product update. However, as we progressed, we opted to leverage events for this purpose.

Events in Ethereum are a way to emit information from a smart contract. They are traditionally used to facilitate communication with web interfaces, such as returning the result of a smart contract method call. In fact, we are employing events in this manner to get the new batch identifier after a `newBatch` call. We have to wait for the event to be emitted since the result of the method call is only available after the transaction has been validated and included in a block.

Additionally, they provide a cost-effective alternative to direct on-chain storage [34]. In a storage setting, events are typically referred to as "logs". According to the Ethereum specification [20], placing 32 bytes in the log consumes 256 Gas units[11], compared to the significantly higher cost of 20,000 units when placed on smart contract storage.

Each event generated during the validation of a transaction is recorded in the block log, including the data associated with the event. The following code snippet defines the event used for each traceability update:

[11] Additional 375 Gas units for LOG operation and 375 for each topic included, although not significant when considering large amounts of data.

```
event Update(
    uint256 indexed batchId, address indexed owner,
    uint48 ts, string documentURI
);
```

These events, however, are not accessible by the smart contracts, meaning they are only useful to transmit data that is not required at runtime.

However, it is important to acknowledge that the event-based approach, while considerably more cost-effective (as observed in Section 4.3.1), is not without its shortcomings. Presently, all logs and events are retained, but there is a potential concern regarding the blockchain's ever-expanding size in the future. To address this issue, the Ethereum networks might implement pruning techniques to optimize storage space, which could lead to the loss of historical event data.

To ensure the long-term integrity and availability of this additional information, we consider storing on-chain storage as a more robust and future-proof approach. Nevertheless, because we initially regarded this data as supplementary, the ramifications of its potential loss are not significant. All the strictly necessary traceability information is stored on-chain as the transaction type. As a result, we have opted to adopt the event-based solution due to its cost-efficiency.

3.2.2 User Registry

The User Registry is an additional component of our system designed to manage system users, specifically members, and actors. This independent smart contract is also tasked with system access control.

Members are exclusively added through a governance process, guaranteeing a controlled and authorized user base. In contrast, actors have an unrestricted registration process and can be added freely. However, actors are only permitted to interact with traceability smart contracts under the ownership of a member after their explicit approval.

Furthermore, the User Registry serves as the repository for supplementary data concerning both actors and members. This encompasses a range of information, such as yearly reports on environmental considerations, accessible via a URI that links to the corresponding data stored in our off-chain storage system.

Like any other component in our system, it is important to design the access control system efficiently to minimize Gas consumption. This is especially important because each interaction with the traceability contract requires an initial authorization check.

Access control models are commonly categorized into two classes: those utilizing capabilities, which grant rights to subjects for specific objects, and those using access control lists (ACLs), which define the list of subjects authorized to access a particular object. In our implementation, a capability list corre-

sponds to a list of authorized traceability contract addresses for each actor, whereas an ACL represents a list of authorized actors for each contract address.

Access control models can generally be categorized into two classes: those based on capabilities and those that rely on access control lists (ACLs). During our implementation, we carefully considered both approaches. Specifically, the Capability List is a subject-oriented list that defines the rights each subject possesses for every object, while the Access Control List (ACL) is object-oriented, specifying the list of subjects with access to a particular object. In the context of our system, a capability list consists of authorized traceability contract addresses stored for each actor, and an ACL comprises a list of authorized actors for each contract address.

However, implementing an ACL would require an additional storage mapping for all the traceability contracts. On the other hand, utilizing a capability list allows us to utilize the existing structure already in place for storing actor-related information. Furthermore, our rationale is that each traceability contract is likely to have more actors interacting than actors will interact in different contracts. This implies that the list of addresses where the actor has been authorized to transact will be smaller, and as a result, it will require less Gas to iterate over the array while checking for specific access.

For the above reasons, we concluded that the capability list offers a more efficient solution and thus opted for its adoption. To implement this, we maintain an array of participating contracts associated with each actor. When an actor is authorized in a new traceability contract, the address of that contract is added to the array. This allows us to easily check if an actor is authorized to transact in a specific contract by searching for the contract's address in the array.

The User Registry is an integral part of our system. Beyond its role in access control, it plays a significant role within our DAO by storing the voting power assigned to each member. Additionally, it acts as a vital component for our interfaces, enabling us to bridge between Ethereum addresses recorded in each transaction, and the actual actor names when presenting information to the client.

3.2.3 Decentralized Autonomous Organization

Our DAO implementation is built upon the Governance contracts from the OpenZeppelin[12] smart contract suite, which we have selected for its efficiency and security. OpenZeppelin is a big reference in the Ethereum ecosystem, for their secure and efficient contracts which have been vetted by talented members of the community. OpenZeppelin's contracts provide a well-established and generalized framework that can be augmented with the use of extensions. However, due to the level of abstraction supported, it means that even the basic set of supported features can sometimes include unnecessary ones for our specific context. We decided to take these governance contracts and tailor them to our implementation in an effort to further optimize the deployment and execution costs.

[12] `https://www.openzeppelin.com/`

In Section 4.4 we dig deeper on the optimization decisions taken. The most significant choices include the removal of the timelock, which imposed mandatory delays on each governance operation, and the integration of the DAO with our User Registry for member voting power, eliminating the need for a specialized ERC-20 token for voting power distribution. Additionally, numerous other minor optimizations have been applied.

As a result of these optimizations, we have achieved a 71% reduction in deployment costs and a 50% decrease in execution costs when considering a typical governance workflow.

Our DAO implementation is comprised of three fundamental methods: propose, vote, and execute. The process is as follows:

- Propose: A member initiates a proposal, specifying its details. These proposals can encompass various decisions, including adjustments to the system's parameters, contract upgrades, or even decisions related to adding or removing members. Technically this decision can be any smart contract method call. Additionally, a description is included in the proposal, describing its purpose to other members.
- Vote: Other members participate by casting their votes in favor or against the proposal. Each member's voting power is associated with their account and is managed within the User Registry.
- Execute: When a proposal accumulates a predefined quorum of votes in favor, it becomes executable. At this point, anyone can trigger the execution, and the method calls included in the proposal will be executed with the governance contract as the calling address.

As referenced earlier, voting power is associated with members in the User Registry. To add a new member to the DAO, they must go through the governance process. This process involves initiating a proposal to add a new member and specifying their voting power. Other members then evaluate the proposal to verify the legitimacy of the new member and assess whether the proposed voting power aligns with the system's requirements.

A fundamental element of our workflow is the requirement that certain contracts can only be modified following a governance process. This access control is executed at the level of smart contract method calls. The most common form of access control involves the concept of ownership: a specific account is designated as the owner of a contract and is thereby granted administrative privileges.

In this approach, the transaction sender's address is compared with the stored owner's address. This ownership model is most adequate for contracts with a single administrative user, whereas when supporting multiple users it is more common to employ a role-based model. However, in our system, the single administrator is not a regular Ethereum account (Externally Owned Account, EOA). Instead, it is the address of our DAO contract, effectively shifting the administration of contracts to a governance-based model.

Finally, in our effort to optimize Gas consumption and accommodate more comprehensive proposal descriptions, including files and images, we have integrated the DAO proposals with our off-chain storage system.

3.2.4 Off-chain Storage

In the process of defining our off-chain storage solution, we explored various alternatives. Initially, we contemplated the idea of enabling actors to utilize their cloud storage providers for data storage. This approach would have involved actors uploading their data to their contracted cloud storage and subsequently submitting on-chain storage a URL to access this data. However, we encountered several challenges with this approach:

- Data mutability: The data's integrity could be compromised after submitting the URL to the blockchain. Any changes or updates to the data might not be accurately reflected in the URL.
- Access control and revocation: Actors could revoke access to their data after a certain period, or if they chose to switch to a different cloud provider, all would lose access to the older data.

As a result, these challenges led us to reevaluate our approach, with a focus on a particular type of storage: content-addressable storage.

Content-addressable storage is a storage method where data is accessed and identified by its unique content-based identifier, typically hash-based, ensuring that any changes to the data result in a different identifier. This approach offers data immutability and integrity, addressing the challenges faced with traditional cloud storage.

Moreover, we aimed to provide an all-in-one interface, allowing users to upload their document data directly within the interface. We reason that this approach could lead to enhanced adoption, as it does not require that users manage their own cloud storage, and grants us more control over how data is stored and displayed.

For data uploaded directly within the interface, we introduced an associated metadata structure to facilitate efficient retrieval. This metadata structure includes the following attributes:

```
{
    "desc": "Example update description", "imgs": ["img1.jpg"], "vids": ["vid1.mp4"],
    "txt": [], "pdf": [], "other": []
}
```

The "desc" attribute holds a description string associated with the upload operation, such as a traceability update. The remaining keys reference lists of file names of that specific file type uploaded alongside the metadata. This detailed association of file names with their respective types enables more

efficient usage of off-chain storage in the web interfaces, allowing for selective downloading of data types we want to display, primarily images and videos, while offering a placeholder to download the other files directly.

The next approaches we considered presented a range of opportunities and challenges:

- InterPlanetary File System (IPFS) [35]: IPFS is a peer-to-peer, content-addressable distributed file system, known for its decentralized nature and large network of interconnected nodes. This technology is widely adopted within web3 projects and is recognized for its maturity in the field.

  IPFS takes an uploaded file or directory and divides it into fixed-sized data chunks, each of which is content-addressable via Content Identifier (CID), an identifier computed from the content's hash. It leverages Kademlia, a Distributed Hash Table (DHT) that identifies the peers containing specific data chunks.

  The initial upload of data to IPFS typically involves saving it to a single peer, and as other peers request this data, a copy is created in their memory, caching for subsequent access. For the data to be accessible within the network, the node has to "pin" the data. Pinning allows the node to advertise that it possesses that CID by linking the corresponding DHT record and the node's IP.

  As part of the retrieval process, the IPFS node checks the integrity of the stored data it fetches by hashing them and verifying that the resulting hash is valid for that CID. This retrieval process is trustless, meaning that blocks can be sourced from any node in the network, and their integrity can be independently verified.

  A noteworthy feature of IPFS is that in its content-addressable storage, where the ID is based on the content's hash, the same file cannot be referenced twice in the DHT resulting in a space-efficient solution. Meaning that subsequent uploads of the same data, will not create a new entry in the DHT instead, it will reference the previously uploaded data.

- Filecoin [36]: When using IPFS there is no direct usage cost. Users can either rent a gateway service, paying a monthly fee for a specific bandwidth, or opt to deploy and manage their IPFS node directly. Another option is the use of Filecoin, which effectively complements IPFS with an incentive layer, preserving its content-addressable nature.

  Filecoin operates as a blockchain-based distributed storage network, where miners contribute storage capacity and receive Filecoin cryptocurrency (FIL) as a reward for providing cryptographic proofs of fulfilling their storage commitments. FIL tokens are exchanged for read and write operations across the network, with all transactions recorded on the Filecoin blockchain ledger.

  Essentially, Filecoin functions as a marketplace for storage providers, incentivizing data caching by rewarding those who maintain the data locally. It is an open system that allows anyone to

participate, to either store their files or get paid for storing other users' files, offering storage and retrieval services in a decentralized manner. Meaning that the available storage, and the price of that storage, are not controlled by any single company.

Security is upheld through a range of proof mechanisms, including Proof-of-Storage, where miners validate their available space and stored data, Proof-of-Replication to confirm unique data copies, and Proof-of-Spacetime to ensure continuous data storage throughout contracted time. As of October 2023, the network proudly exhibits an impressive total data storage of approximately 1.5 EiB. However, the need for consumers to pay a fee for content retrieval is a drawback that led us to exclude this option, as it does not align with our requirement that consumers should have free access to the data they wish to inspect.

- Swarm [37]: Ethereum Swarm is similar to Filecoin and IPFS in terms of being content-addressable and decentralized, but stands out for its tight integration with Ethereum smart contracts and built-in incentive layer. This incentive system operates through smart contracts on the Ethereum blockchain and is powered by the BZZ token.

  In this system, like the previous ones, a DHT is employed, but it takes a different approach. Instead of referencing the address of the peer containing the data chunk, Swarm stores the chunk directly in the DHT, an approach known as DISC (Distributed Immutable Store of Chunks).

  Anonymity stands as a key feature within the Swarm network. Each node is only aware of a limited number of its immediate neighbors. If a chunk is not found among these neighbors, they request subsequent nodes and forward back the content, thus obfuscating the identity of the requestor and the source of the given chunk.

  When retrieving a data chunk in Swarm, a fee is paid in BZZ tokens, similar to the Gas costs on the Ethereum network. If the chunk is present with an immediate neighbor, the requesting peer retains the BZZ fee. However, if the chunk is not found locally, it needs to compensate one of its neighbors for the requested data. This setup creates an incentive for nodes to cache frequently requested content and ensure its proximity to users, effectively transforming Swarm into a highly efficient content delivery network.

  Nevertheless, mirroring the situation with Filecoin, the necessity of charging a fee for data retrieval conflicts with the requirement of providing free access to end-users.

- BitTorrent: BitTorrent is a well-established peer-to-peer file-sharing protocol that differs from the previous content-addressable options. It is mainly used to reduce the impact of distributing large files, as it downloads small pieces of the file from different peers, simultaneously. The integrity of each piece is provided by a cryptographic hash stored in a file alongside the URL for the tracker. A tracker, in this context, is a server that monitors and manages active peers possessing these file

pieces. It helps facilitate the sharing process by connecting downloaders (leechers) with uploaders (seeders) who have the required pieces.

The file containing this metadata used to reference the uploaded file is called “Torrent file” and is part of the reason we avoided the use of this solution, as it would require significantly more on-chain storage compared to storing a content ID of a content-addressable storage. Additionally, we reasoned that there would be no incentive for other peers to replicate our data and so we would not benefit from this approach.

Out of the various off-chain storage solutions we considered, we decided to implement our system using IPFS directly. During the development phase, we established a local instance of Kubo, the default IPFS implementation written in Go.

Managing our dedicated IPFS node would afford us a high degree of control over data access and availability. However, our system also supports the use of a gateway provider, e.g. Infura[13], with minimal configuration changes. The use of a gateway service could offer significant advantages in a production environment, providing a potentially more stable and lower-latency network.

Regardless of whether we are using an IPFS node directly within a resource-constrained system or a gateway service with restricted bandwidth, we considered the implementation of access control essential for writing operations. Without such controls, there is a risk that anyone could potentially exploit our system to upload arbitrary data to the IPFS network.

Efforts have been made to create a modified version of IPFS that integrates with Ethereum smart contracts to enable access-controlled file sharing [38]. In our case, we opted for a simplified approach that did not require any modifications to the IPFS codebase. Instead, we established the sole connection to our IPFS node or gateway service through an API endpoint. This endpoint serves as a filter, ensuring that only authorized parties are permitted to make upload requests.

To ascertain whether a user is part of our system and has the authorization to publish data, we queried the existing access control mechanisms in the User Registry smart contract. Our verification process entails the inclusion of a digital signature in the payload sent to the API endpoint. This signature is generated using the user’s Ethereum account’s private key and is computed locally using wallet software ”eth_sign” or ”personal_sign” operations.

When signing arbitrary data we use an Ethereum-specific signature using the formula:

```
sign(keccak256("\x19Ethereum Signed Message:\n" + len(message) + message)).
```

By prefixing the message in this manner, the resulting signature is uniquely identifiable as an Ethereum-specific signature. This precautionary measure prevents some potential misuse cases where a malicious application could leverage wallet software to sign arbitrary data (e.g., a transaction) and employ the signature to impersonate a legitimate user.

[13] https://www.infura.io/

The message sent to the API endpoint consists of the following parts:

```
[file_contents, description, contractAddress, batchId, timestamp, digitalSignature]
```

The digital signature is computed over the keccak256 hash of the remaining parts.

Verification of this signature, followed by the utilization of the “ecrecover” algorithm, allows the API endpoint to determine the public key based on a given message digest and the signature generated by the private key for that specific digest. With access to the public key, we can compute the corresponding Ethereum address, which is then cross-referenced with the data stored in the User Registry smart contract to determine whether the actor is authorized to operate within that smart contract.

To enhance security and prevent potential replay attacks, we consider the addition of timestamp validation. While it may not pose an issue due to the idempotent nature of the operation (where reuploading the same files that are already stored would not cause harm), timestamp validation safeguards against potential attempts of server resource starvation. We validate the timestamp by ensuring that it falls within an acceptable time window and is earlier than the last recorded update for that specific batch. To ensure our system maintains a consistent clock, the timestamps used are queried from the Ethereum provider’s block timestamps.

One of the key aspects of our solution is the use of URI for both data access and integrity verification, a feature that notably reduces the on-chain storage demands. In our off-chain storage approach, we record the URI of the data on-chain to serve as a reference for later access. Within the context of IPFS, this URI is known as CID.

Each piece of content is identified by a unique CID, generated from the cryptographic hash of the content itself. This hash is specifically derived using a SHA-256 digest of the published data, combined with additional metadata such as the CID version. The CID serves a dual purpose: it allows for direct content retrieval, and acts as a means of maintaining data integrity by comparing the hash of the retrieved data with the hash included in the CID, ensuring that the content remains authentic.

CIDs come in various forms with different encoding bases and CID versions. For example, a CID version 0 consists of 46 characters, while a version 2 CID has a length of 59 characters. To ensure future-proofing, we made the deliberate choice to support both as well as any upcoming versions. This flexibility is achieved by storing CIDs in a dynamic string format rather than a more cost-effective, but less versatile, constant-length byte array. This approach ensures our system’s compatibility with evolving CID standards and variations.

In Section 4.3.1 of our evaluation we analyze the cost benefits of employing an off-chain storage approach.

In our final decision, we chose to keep all data on IPFS unencrypted. This choice aligns with our initial idea, where the information should be publicly accessible for both the client and all users within

our interface, thus maintaining the transparency of our system.

Figure 3.6 depicts the ”all-in-one” user interface to upload additional documents to the off-chain storage.

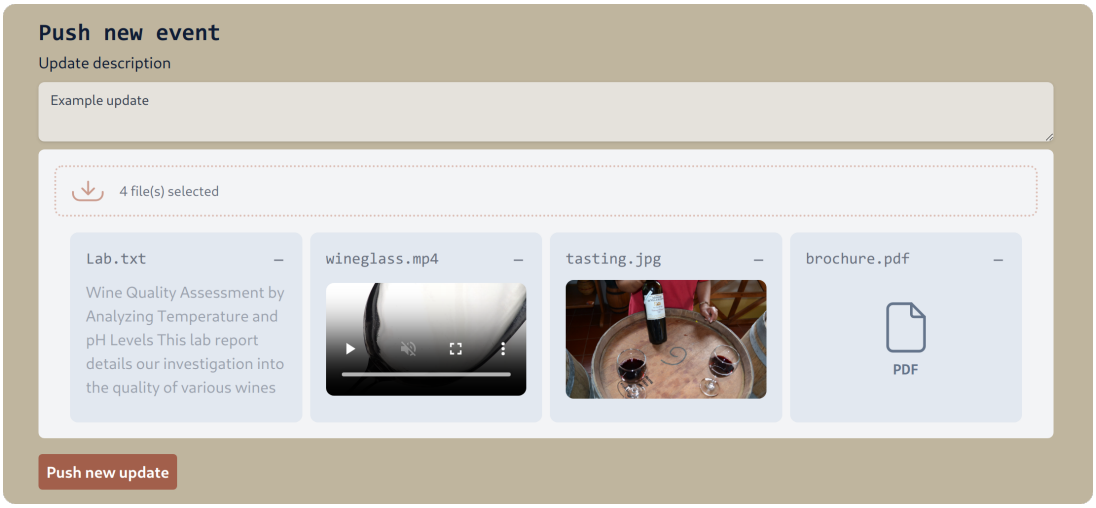


Figure 3.6: Interface for uploading additional documents to a traceability update

3.2.5 Product Identification

We reasoned that incorporating QR codes into our system was necessary as it facilitates user interaction with our system.

To eliminate the need for multiple product codes on our items, we aim to establish a unified identifier for both retail checkout and batch traceability within our system. To achieve this, we adopted the GS1 Digital Link standard [39]. GS1 is an international standards organization renowned for introducing barcodes and managing product identifiers, such as GTINs (Global Trade Item Numbers).

The GS1 Digital Link standard is designed to create web-enabled barcodes and QR codes capable of carrying comprehensive product information. These codes serve a dual purpose by functioning as both traditional barcodes and web-based URLs. Depending on the scanning system employed, they can reveal different data aspects, delivering point-of-sale functionality and enhancing the consumer experience.

In particular, when a point-of-sale scanner is used to scan the digital link QR Code at a store’s checkout, it behaves like a conventional barcode, reading the GTIN associated. However, when consumers scan the same code on the product’s packaging with their mobile devices, it redirects them to a web page containing relevant content, in this case, Tracer’s client interface.

At the core of GS1 Digital Link is a standardized URI syntax designed to serve various applications. This syntax defines specific identification keys that categorize the URI into distinct parts. Notably, “01”

is the key used to identify the GTIN, while “10” is designated for the Batch/Lot identifier. These keys are employed in the following manner:

As an example, the URI structure for GTIN 9520123456788 combined with Batch/Lot “wCR9...” in a valid GS1 Digital Link URI appears as follows (GTIN + Batch/Lot):

```
https://CLIENT.INTERFACE
    /01/9520123456788
    /10/wCR9E01fiMPWq1VCqPaSHIwjprRna4snApe1yK9DfhM@nxrFS-8N0vbzRi6g-pT8YjAN0o4
```

In our system, we employ the GS1 Digital Link Batch/Lot identifier as the primary ID for our client interface. This identifier is established by concatenating the batch ID generated by our smart contract with the associated contract address. However, this approach necessitates including a substantial amount of data in our QR codes, which raises some concerns.

The inclusion of the Batch ID and Contract address in the QR code results in a more complex code that may require a larger print size or more precise scanning and printing. Additionally, we added our system’s logo to the QR code as we consider that it could enhance consumer recognition of products employing our system. As a result, we had to increase the error correction level of our QR codes, adding to the QR code complexity even more.

This complexity could be problematic for older devices and situations where fast scanning is expected, such as at checkout terminals. Slow and inefficient scanning might also lead to consumer dissatisfaction and frustration.

One common solution employed when generating QR codes is to use a URL shortener, but this approach would sacrifice the use of GS1 Digital Link and introduce additional overhead into our system, requiring the storage of extra information to match the shortened URLs.

Instead, we chose to encode the batch ID and contract address values in base64 from their binary value. By skipping the number notation altogether, we achieved a size reduction of 41% but at the cost of human readability.

Furthermore, our QR codes are generated in vector image format to allow for easy resizing and modifications, such as color-matching the product packaging, while preserving image resolution. This flexibility is aimed at facilitating the integration of our system with product packaging, making it more convenient for product companies to incorporate our solution.

Figure 3.7 depicts an example QR code generated by our system.

3.2.6 Web Interfaces

The user interfaces were implemented in a web-based model. The connection between the application and users’ Ethereum accounts is established by a wallet, e.g. Metamask.

Figure 3.7: Example QR code generated by Tracer

In our consumer interface, we have implemented server-side rendering, aiming to reduce interface latency. Unlike the conventional client-side workflow, where users request both the HTML template and the script code responsible for querying essential data, our server-side approach speeds up this process. In this setup, the server itself requests the data and provides the page with the data already incorporated, effectively eliminating the latency associated with a roundtrip.

Moreover, this architecture enables finer control over the latency of Ethereum nodes. Looking ahead, should we decide to deploy a full Ethereum node for our application, we can establish a high-speed connection between the node and the front-end server, further improving the overall system latency.

To ensure a clear separation between our user interface and the underlying system, we have employed the use of an API file. This file serves as an intermediary, providing the necessary operations, and abstracting interactions with the blockchain.

With the help of a tool known as TypeChain[14], we have generated TypeScript types for our smart contracts. These types are designed for seamless integration with the Ethers[15] library, which we rely on for interacting with the Ethereum blockchain. However, to further reinforce the separation between our interfaces and the underlying data representation, we have taken the additional step of creating intermediate types, such as “Batch” or “Update”. This abstraction allows us to smoothly transition between different libraries or even adapt to changes in the underlying blockchain technology. For instance, should we decide to shift from Ethers to another library like Web3.js or explore a change in the blockchain infrastructure, the process would entail adjustments primarily within the API itself, without extensive refactoring of the interface code.

Our interfaces source data from the off-chain storage. Updates encompass various document types, including images and videos, which are displayed directly. For other file types, users have the option to download and subsequently view the documents locally.

[14] `https://github.com/dethcrypto/TypeChain`
[15] `https://docs.ethers.org/`

4

Evaluation

Contents

In this chapter, we will assess the performance and efficiency of our system using two metrics: latency and execution cost. Our evaluation will focus on the blockchain aspects (smart contracts) rather than the interfaces.

In this context, we define latency as the time it takes to complete an operation within our system, i.e. the time since the transaction is sent to the network until it has been successfully added to a valid block, in the case of an operation that mutates the blockchain state, or until a response with the requested value is received, for viewing operations.

Execution cost refers to the amount of Gas consumed by the EVM for each operation within our system.

These metrics have a direct impact on our system's users. Latency, in particular, is important to the

end consumer as it directly influences the user experience with the customer interface. This interface will query batch-related data, and any delays can affect the user's experience.

Execution costs, on the other hand, do not directly affect customers since there are no fees associated with using the consumer interface. However, these costs can have an indirect impact as actors within the system are responsible for covering these fees. This could potentially result in an increase in product prices, affecting the end consumer.

Our evaluation was conducted through a suite of tests developed in the Hardhat[1] development environment, and we utilized the Ether.js[2] library for deploying and interacting with the Sepolia test network. Further details about test networks will be provided in the following section.

Throughout our evaluation, we assumed constant prices. For Gas costs, we used the average Gas price as of September 23, 2023, which was an average of 15 gwei per Gas unit. Additionally, to account for the volatility of Ethereum prices, we employed the September month's average price of 1540 euros per ETH as a reference.

4.1 Deployment

Our system's smart contracts were developed in the Solidity programming language which can be compiled to EVM executable bytecode, using the `solc` compiler. During design and development, we focused on supporting the Ethereum network but being compiled in EVM bytecode means that our system could be deployed in other EVM-compatible blockchains.

The Mainnet is the primary Ethereum production blockchain, and for cost comparisons, we will be assuming Mainnet costs[3]. In addition to the Mainnet, there are public testnets. These are mainly used by developers to test their smart contracts in a production-like environment before deploying to the Mainnet. On test networks, it is still necessary to own ETH to interact with Ethereum but it can be acquired for free from faucets and have no real values. Faucets are, typically, web applications that send ETH to a specified Ethereum account.

There have been several test nets, with differentiating aspects, but after the Ethereum Merge[4] the currently active ones are:

- **Goerli[5]:** Goerli was the first testnet utilizing proof-of-authority consensus and was designed to be compatible with different Ethereum client implementations. Recently after the Ethereum Merge it moved to proof-of-stake consensus with an open validator set, meaning that everyone is able to run and test a validator, i.e. a node, for Goerli. It has been deprecated as of January 2023, but it

[1] `https://hardhat.org/`
[2] `https://docs.ethers.org/v6/`
[3] `https://ethereum.org/en/developers/docs/networks/`
[4] `https://ethereum.org/en/roadmap/merge/`
[5] `https://goerli.net/`

is still used for testing protocol upgrades before these are deployed to the Mainnet. The network was started in 2018 and has a rather large size for a testnet meaning that it requires more space and a long time to sync when starting a new node. Another downside is the hard cap on the total amount of Goerli ETH, which imposes a necessary throttling to Faucets, like limiting the amount of ETH that can requested each day. This is a reason why the network has been deprecated and developers are moving to newer ones.

- **Sepolia**[6]**:** It is a permissioned proof-of-stake testnet that quickly became the recommended default testnet for application development after the Merge. While developers can use this network to deploy and test their systems, the permissioned setting of this testnet means that it uses a closed validator set, where you need to request permission to run a validator node and not everyone can opt-in. Being fairly new also means that the state and history are quite small, translating into quicker sync times and less required storage space for new nodes. Unlike Goerli, which has a hard cap on ETH tokens, this testnet offers an unlimited supply, making it the preferred choice for developers.

- **Holešky**[7]**:** Holesky, or Holešovice, is set to replace the now deprecated Goerli. It is a permissionless proof-of-stake network, similar to Goerli, and will replace it as a staking, infrastructure, and protocol-development testnet. The goal with Holesky will be to support a large number of validators, larger than Ethereum Mainnet, to catch possible scaling issues of the protocol. It will also launch with a larger initial supply of ETH tokens to mitigate some of the complaints about Goerli.

From the available test networks, our choice for evaluation purposes will be Sepolia. This network closely emulates the configuration of the Mainnet and is anticipated to be more stable due to its closed validator set. Its popularity and ease of use, especially when retrieving ETH tokens, also greatly influence the decision.

Table 4.1: Deployment cost of our solution's smart contracts

Contract Name	Gas Used (gas)	Deployment Cost (euro)
Traceability Contract Factory	1,814,993	**41.93**
Governor Contract	1,666,696	**38.50**
User Registry	1,301,895	**30.38**
Executor	671,286	**15.51**
Total	5,468,064	**126.31**

Table 4.1 depicts the cost of deploying the smart contracts from our system. The cost is defined in Gas units and is directly proportional to the size of our contracts. It is a one-time cost associated with the deployment transaction and calculated based on the fee for each Gas unit, 15 gwei per Gas unit in our evaluation.

[6] https://sepolia.dev/
[7] https://github.com/eth-clients/holesky

To recap the purpose of each contract: The Traceability Contract Factory contract is used to create new instances of our traceability smart contract; The Governor Contract is responsible for the governance process in the DAO; Both the User Registry and Executor are additional contracts of the DAO, serving to store member information and act as an intermediary layer for executing proposals, respectively.

While the upfront cost of 126 euros may initially appear steep, it is important to recognize that this expense represents a premium for the permanent recording of our data on the blockchain over thousands of nodes. Moreover, this cost has significantly decreased compared to the period before The Merge, e.g. in November 2021 the deployment of our smart contracts to the Mainnet would cost more than 3000 euros. However, as discussed in Section 2.2, the substantial value and particularly the high execution cost have prompted developers to explore more cost-effective alternatives such as Layer 2 solutions and sidechains. For instance, on the Polygon[8] sidechain, the total deployment cost amounts to approximately 0.34 euros, while on the Arbitrum One[9] Layer 2 chain, it comes to around 0.83 euros.

It is important to note that the previous values were obtained when deploying our smart contracts after compiler optimizations. The solidity compiler, solc, is used to compile smart contract code written in the Solidity programming language, version 0.8.19, into EVM-compatible bytecode. This compiler is equipped with an optimizer that tries to simplify complicated expressions, reducing both code size and execution cost - Gas.

The optimizer configuration used is depicted in Listing 4.1.

Listing 4.1: Solidity compiler configuration

```
solidity: {
    version: "0.8.19",
    settings: {
        optimizer: {
            enabled: true,
            runs: 1000,
        },
    },
}
```

The runs parameter, is an option of the solidity optimizer[10] that roughly specifies how often each opcode of the deployed code will be executed. As explained in the official Solidity documentation, the value for this parameter is a trade-off between deployment cost and execution cost. A value of “1” results

[8] https://polygon.technology/
[9] https://arbitrum.io/
[10] https://docs.soliditylang.org/en/latest/internals/optimizer.html#optimizer-parameter-runs

in concise but costly code, while a higher value generates longer but more Gas-efficient code. We chose a higher value since our smart contracts are predicted to have a high influx of transactions, preferring cheaper transactions over deployment costs.

4.2 Latency

The latency evaluation was performed on September 23, 2023, using Infura[11], a service provider for accessing Ethereum nodes, to connect to the Sepolia test network.

Table 4.2: Latency measurements for Sepolia network

Metric	Value (s)
Average Block Time	12.6
Average Network Latency	0.188

Before we evaluate our solution in terms of latency, and especially since we are running on a test network, whose performance is not as stable or consistent as the Ethereum Mainnet, we will first programmatically assess the network state at the time of our evaluation. We present two metrics in Table 4.2: average block time and average network latency.

The average block time represents the time taken for a new block to be included in the blockchain. We measure it by taking the latest 500 blocks and averaging the timestamp difference over the number of blocks. By considering only the latest blocks we better represent the current network state, avoiding older periods of higher latency. For example, when averaging the latest 1 million, the block time is approximately 13.18 seconds compared to the faster 12.6 seconds when querying only the latest 500 blocks.

The average network latency was measured from 25 successive Remote Procedure Call (RPC) to the node method `getBlockNumber`, which returns a small payload containing the current block number. We observed an average value of 188.37 ms, This result largely depends on the backend infrastructure we are using, e.g. node geographic location, and our connection to it, and will be considered constant during our testing.

Following the fundamental network metrics, we can proceed to draw some conclusions from our system latency test results.

It is important to note that on EVM-based blockchains, the computational effort is not quantified in execution time, it is instead measured in Gas consumption. In fact, for our evaluation, the execution time is not nearly as important because a transaction, i.e. our smart contract method call, is only considered complete and the new state visible to others when the block containing that transaction is included in the

[11] `https://www.infura.io/`

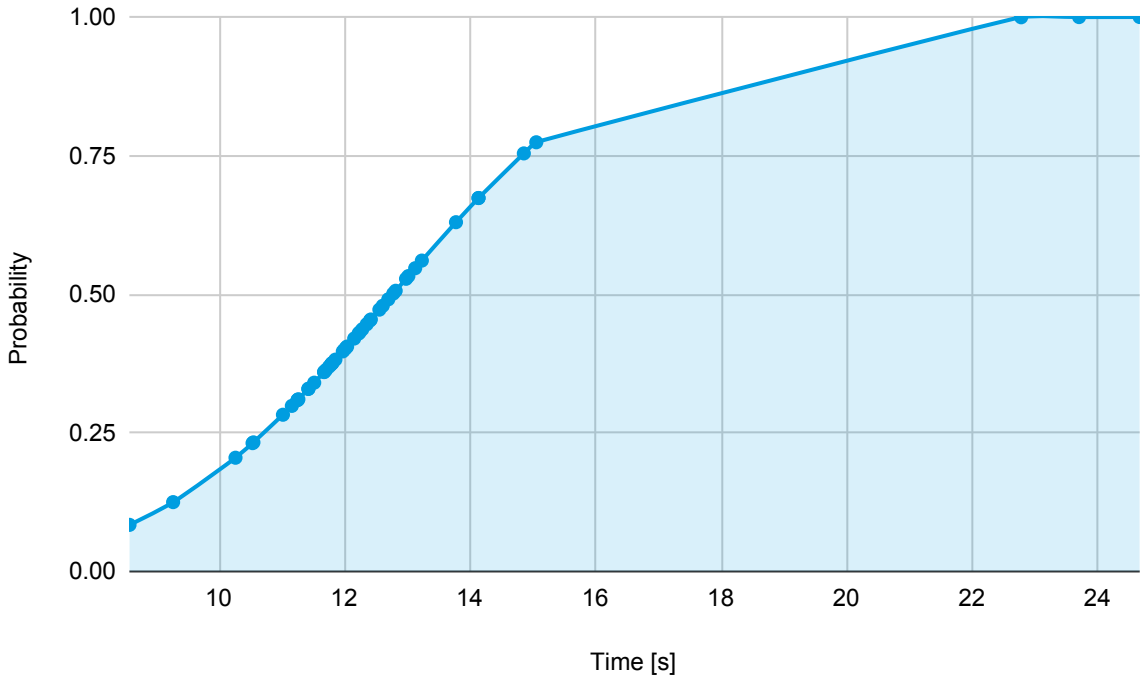


Figure 4.1: Cumulative distribution for latency values of state-changing transaction (`handleUpdate` method) in Sepolia network

blockchain. So we will be considering the block time as the latency when interacting with our system's smart contracts.

In Figure 4.1, we present the latency distribution of 50 state-altering transactions. In this specific scenario, the transactions involve adding a new update, i.e. additional information about a product, to our traceability smart contract via the `handleUpdate` method call. The observed latency is 12.764 seconds, which is similar to the average block time of 12.6 seconds. This similarity confirms that our transactions do not affect the block time and that we can assume that the latency when interacting with our system is approximately the same as the average block time. Therefore we are able to infer the latency on other EVM networks by comparing its average block time. For example in the Ethereum Mainnet, we would get an approximate execution time of 12.08 seconds, the current block time for that network.

There are, however, some discrepancies from some outliers that can be attributed to possible network instability or an unusually high amount of transactions pending in the *mempool* – transactions that are waiting to be confirmed and added to a block.

One exception to the earlier assumption regarding state-changing transactions is the case of read-only operations, as demonstrated in Figure 4.2. These operations are significantly faster, averaging 179.91 ms. This result is based on our testing, which involved 50 calls to the `getBatch` method, respon-

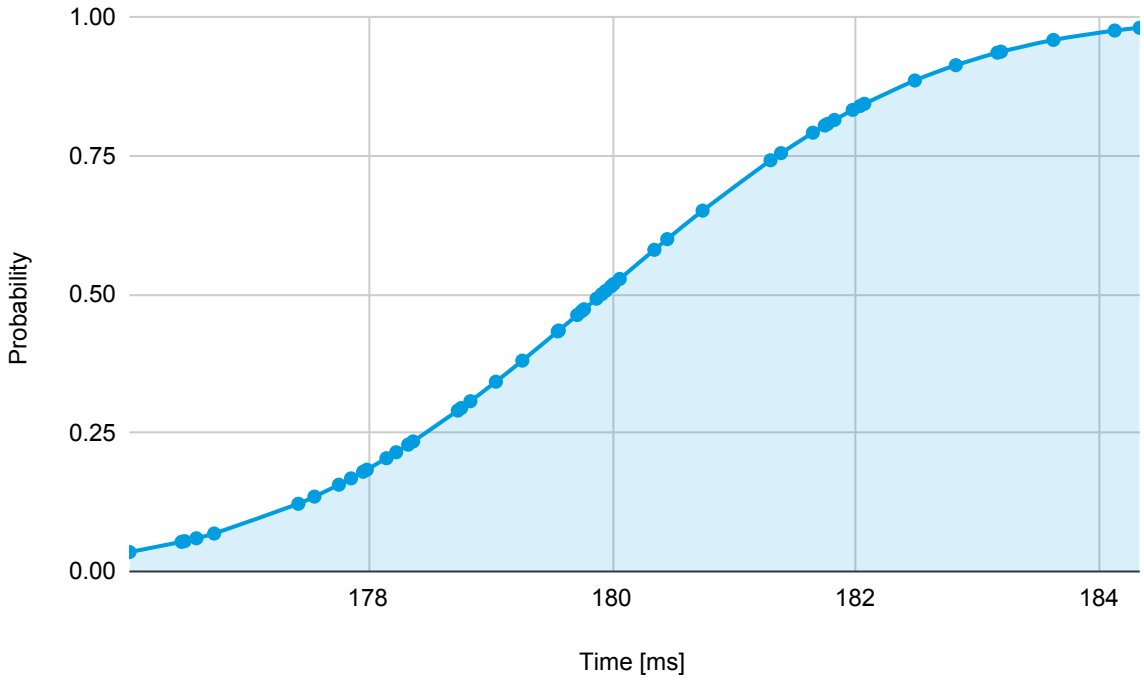


Figure 4.2: Cumulative distribution for the latency of blockchain state query (`getBatch` method) in Sepolia network

sible for retrieving a specific stored batch structure.

In fact, requesting the current blockchain state, such as retrieving information from smart contract storage, can be approximated to a database query, which is rather fast on the infrastructure used since the average latency is similar to network latency, so the operation overhead can be regarded as negligible and the overall latency approximated to the network latency.

In our system interface, before displaying the information to the user, a `getBatch` query will be executed hence the performance of read operations will directly influence the usability and client satisfaction. Therefore it is important to evaluate and utilize an infrastructure provider with low latency or ultimately manage our own infrastructure.

4.3 Execution Cost

This section will evaluate our system regarding the execution cost while interacting with our smart contracts. The cost is measured in Gas. Gas refers to the unit that measures the amount of computational effort required to execute specific operations on the EVM. As referenced before, we will be assuming a Gas cost of 15 gwei ($1.5 * 10^{-8}$ ETH) and 1540 euros per ETH token.

We begin by evaluating the cost of individual methods, accomplished through unit tests that record

the Gas consumption for each operation. This assessment provides us with valuable insights into both the cost-effectiveness of our solution and the efficiency of our individual methods, given that the Gas used directly represents the necessary expenditure of computational resources. A more efficient solution translates into less Gas expended. Examine Table 4.3 for the obtained values.

Table 4.3: Execution cost of our smart contracts individual methods

Contract Name	Method Name	Gas Used (gas)	Execution Cost (euro)
Traceability	`handleTransaction`	86,323	2.00
Traceability	`handleUpdate`	44,079	1.02
Traceability	`newBatch`	156,063	3.60
Traceability	`changeConformityState`	32,252	0.74
TraceabilityContractFactory	`create`	197,443	4.56
GovernorContract	`propose`	90,382	2.09
GovernorContract	`vote`	86,854	2.01
GovernorContract	`execute`	22,010	0.51
UserRegistry	`addActor`	136,181	3.15
UserRegistry	`addMember`	160,870	3.72
UserRegistry	`updateActor`	39,265	0.91
UserRegistry	`updateMember`	41,315	0.95
UserRegistry	`addContractToActor`	68,854	1.59
UserRegistry	`updateActorState`	29,629	0.68

Note that for every argument of type string, we used a constant 31-character value, and where a document URI was required, we used a IPFS CID of 42 characters. A value of 31 characters was chosen due to the fact that the EVM storage works with 32-byte words, so the 31-character string plus required metadata can fit in a single storage word.

When comparing the utilization of the EIP-1167: Minimal Proxy Contract (previously explained in Section 3.2.1), also known as a clone factory, and a conventional Factory Pattern (which deploys each smart contract directly), we observe significant differences in Gas consumption. The Gas used for deploying and initializing a traceability smart contract the traditional way, i.e. deploying and calling the method directly, would be 1,447,635 Gas units. Compared to the `create` method in Table 4.3, which performs the exact same logic but through a proxy contract that creates a clone of an already deployed smart contract, the Gas used is merely 197,443 units. By implementing the clone factory, we achieved a noteworthy reduction 86.36% in Gas used when creating a new traceability smart contract.

However, it is important to note that during typical usage of our system, traceability contracts are generated through governance proposals. In our previous discussion, we covered the creation of the actual smart contract, which is part of the cost and the rest is the overhead introduced by the governance process. We will explore these details further in our upcoming workload section (4.3.2).

As for the individual governance methods tested in this section, they were evaluated with a proposal payload for creating a new member. It is important to reference this because the presented values for `propose` and `execute` are affected directly by the payload of their transactions. The `vote` method, on

the other hand, will not fluctuate with the payload since it uses the proposal identifier directly.

This payload encompasses the following components:

1. An Application Binary Interface (ABI) encoded method call for the `addMember` method of the UserRegistry.

2. The address of the UserRegistry contract.

3. A proposal description, comprising an evaluation string limited to 31 characters.

4. The value of ETH tokens to be sent alongside the transaction, which remains at 0 when creating a new member.

The payload serves a crucial role in computing a 'proposalId', an internal reference for the proposal and its associated state. This payload is transmitted twice: first when creating the new proposal to compute the 'proposalId' and second when executing the proposal.

This particular approach, inherited from OpenZeppelin's original contracts, enables the execution of proposals with the necessary parameters (payload) while avoiding the permanent storage of the payload on-chain during the governance process. This strategic choice dramatically reduces execution costs.

To further clarify, the payload from the `execute` method call is employed both for computing the 'proposalId', to validate the proposal state, and directly as the call parameters when executing the proposal. The `vote` call, on the other hand, solely requires the pre-computed 'proposalId'.

Additionally, it is worth highlighting that in Table 4.3, for the `execute` call, we deduce the Gas consumption exclusively for the `execute` method call by subtracting the Gas used by the subsequent `addMember` method call when executing the proposal. For a comprehensive understanding of the overall cost of adding a member through the governance process, please refer to the upcoming workload section (4.3.2).

Our system was envisioned to work for premium products as well as for everyday ordinary goods. However, imposing a 2 euro fee per transaction may appear impractical for less expensive, common products. It is worth noting that our utilization of batches as traceability units helps distribute the cost across all products within a batch, resulting in a substantial reduction in the execution cost per product. Nevertheless, as previously referenced in Section 4.1, the use of layer 2 or sidechains could greatly reduce these fees. On the Polygon sidechain, a transaction would cost 0.0033 euros and an update 0.0017 euros, while on the Arbitrum One Layer 2 chain, the cost would be 0.0139 euros and 0.0071 euros, respectively.

4.3.1 Off-chain Storage

An important feature of our system is its capability to augment the basic traceability data of our products with additional information. We find this additional information advantageous not only to clients, who gain deeper insights into the products they purchase, but also to companies seeking to enhance consumer-brand engagement by offering transparency into their processes.

During the development of our system, we explored several options for managing the storage of this supplementary information. One conventional approach would involve storing the data in a centralized database, allowing authorized actors to push their data to this centralized repository. While this solution would be cost-effective and the data not as critical as the core traceability information, it does not fit in our system requirements since it introduces a degree of centralization and questionable integrity, undermining its reliability.

An alternative, albeit naive, approach within the realm of smart contracts would be to store our document content directly in the smart contract's storage. This approach ensures data integrity and non-repudiation, aligning with our system requirements. However, as we've observed, on-chain operations, especially storage operations, are relatively expensive. Nevertheless, blockchain technology has gained traction in applications demanding significant data storage, such as our system, and developers have been actively seeking ways to maintain the advantages of decentralized applications while addressing the accompanying cost challenges.

Our chosen solution, and a key part of our implementation, involves the use of off-chain storage systems, with IPFS serving as our choice. IPFS is a decentralized peer-to-peer storage system. In this system, content is identified by a unique CID, derived from the cryptographic hash of the content, specifically a SHA-256 digest of the published data, combined with additional metadata like the CID version. The CID is used to query the content directly in the content-addressable storage and is also used to maintain data integrity, as we can verify the data by comparing the hash of the retrieved data with the hash included in the CID.

By storing the CID, which is small and has a constant length of 42 characters regardless of the data's size, we ensure that blockchain transaction costs remain independent of the amount of data actors wish to include. This incentivizes actors to share data without concerns about Gas fees. This solution retains the decentralization of data, maintains data integrity, and incorporates the properties of storing entire contents on-chain, while only now storing just the CID.

We compared both approaches regarding Gas consumption on varying file sizes, on Figure 4.3. Note that the file content was sent as a base-64 encoded string, which can result in a size increase of up to 4/3 of the original data, but we will be considering the original file size for comparison since that is relevant data and discarding the overhead from the base-64 encoding.

Our analysis revealed that Gas consumption using the smart contract storage approach is directly

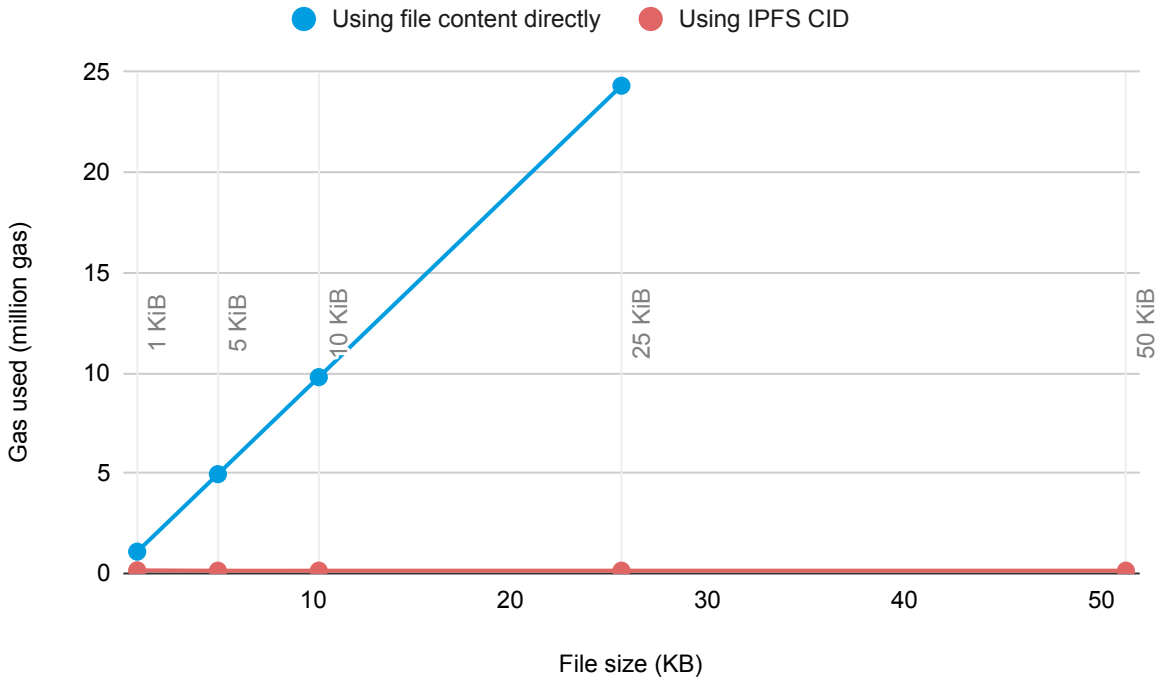


Figure 4.3: Comparison of Gas consumption for different file sizes using smart contract storage approach

proportional to the file size. However, when we employ the off-chain storage solution, represented by the use of CID, we observe a constant Gas cost. This constant cost is because CIDs have a fixed length, regardless of the size of the file they reference.

Concerning the cost of storing data in IPFS, there are no direct charges for end users associated with this operation. The actual costs will vary based on whether we utilize our own IPFS node, which incurs the hardware and operational expenses typical of a storage server. Alternatively, if we choose to use a gateway service such as Pinata, the owner incurs a monthly fee for storage access, which can amount to approximately 0.05 euros per GB monthly.

An important finding from this comparison is that there is a maximum size limit for on-chain storage. Our testing showed that files larger than 100 kilobytes could not be stored on the Ethereum blockchain. Theoretically, there are $2^{256} - 1$ word slots, each with 32 bytes, as available storage for every smart contract on the EVM. However, in practice, we encountered an error before reaching this theoretical limit. This error is influenced by the Gas limit imposed on each transaction within the Ethereum network.

Every Ethereum transaction is restricted by a maximum amount of Gas that can be used within a block, known as the block Gas limit. Typically, this limit hovers around 15 million Gas units but can be adjusted by network validators based on network requirements. Ethereum, however, enforces a maximum block Gas limit of 30 million Gas units to control the rate at which new transactions are processed.

This limitation is essential for maintaining short block times, a crucial factor for applications requiring low-latency network interactions.

Our tests revealed that the Gas consumed during the last successful update, for a file size of 25 kilobytes, approached the maximum block Gas limit of around 30 million Gas units. Consequently, we conclude that we cannot publish larger documents in our system smart contracts storage because the transaction requires more Gas than the current limit.

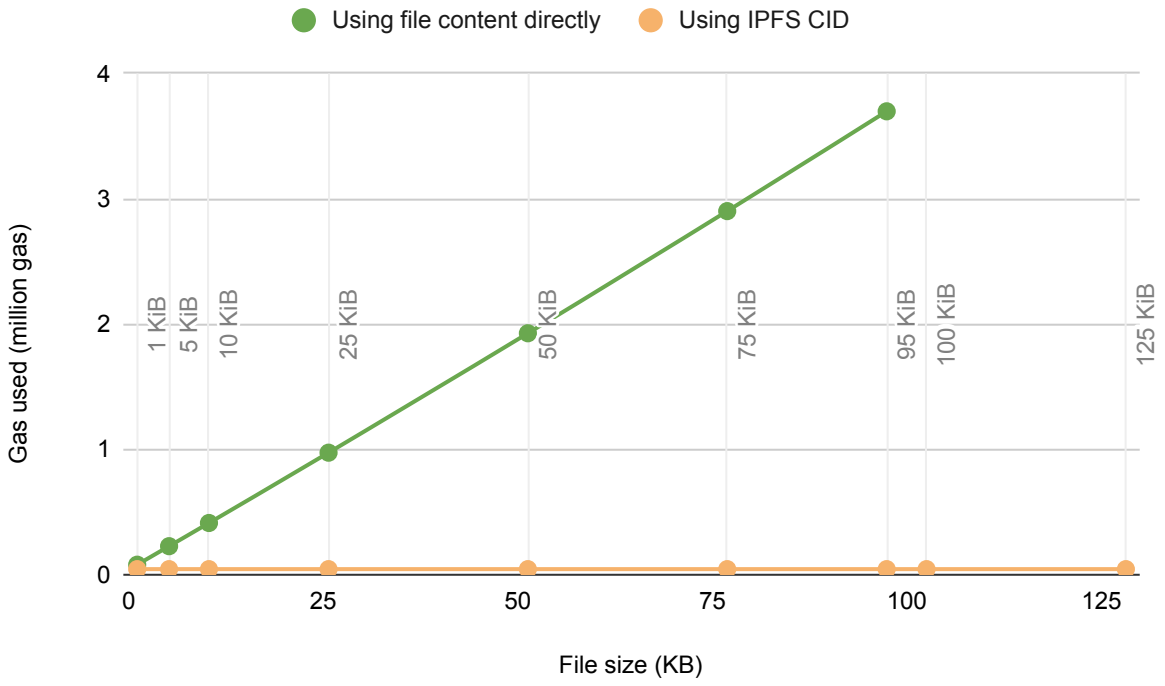


Figure 4.4: Comparison of Gas consumption for different file sizes using smart contract events approach

The next approach tested explored a more cost-effective method for disseminating data on the Ethereum blockchain. Rather than storing the file data directly in the smart contract's storage, we leveraged Ethereum's event mechanism to distribute this data. In Figure 4.4, we compared both scenarios again, publishing the file contents directly and via an off-chain storage system but this time using the event-based approach explained in Section 3.2.1, where we try to leverage Ethereum events as a cost-effective storage alternative.

Although a significant reduction in Gas consumption was observed, we again encountered a maximum file size limitation. In this scenario, documents larger than 100 KiB could not be uploaded. Surprisingly, the Gas used for 95 KiB remained significantly below the block Gas limit, which was the limiting factor in the previous approach. This limitation occurs due to a specification in the Ethereum node implementation employed by validators on the Sepolia network. Most Ethereum nodes run a version of

go-ethereum (`geth`), which sets a configurable parameter for the maximum size for individual transactions allowed in its mempool. The default value for this parameter is 1 MiB, as observed in the source code in Listing 4.2.

Listing 4.2: Go-ethereum source code - maximum transaction size

```
// Source: https://github.com/ethereum/go-ethereum/blob/master/core/txpool/
    blobpool/blobpool.go\#L67
// txMaxSize is the maximum size a single transaction can have, outside
// the included blobs. Since blob transactions are pulled instead of pushed,
// and only a small metadata is kept in ram, the rest is on disk, there is
// no critical limit that should be enforced. Still, capping it to some sane
// limit can never hurt.
txMaxSize = 1024 * 1024
```

During our testing, we discovered that we were unable to publish 100 KiB of file contents. From the error message received, we were able to infer that the validator attempting to execute our transaction had a configured limit of 128 KiB. While this limit may vary in Mainnet validators, it would invariably remain a limiting factor for our system.

In response to the obtained results, we settled on an approach that combines both solutions. We stored the file content in off-chain storage while publishing the CID through smart contract events. This hybrid approach substantially reduced operational costs in the traceability smart contracts. Gas usage decreased from 135,279 for `handleUpdate`, 172,659 for `handleTransaction`, and 264,665 for `newBatch` when using smart contract storage to 44,115 (a 67% reduction), 86,395 (a 50% reduction), and 156,099 (a 41% reduction), respectively. Even the creation cost for a new traceability smart contract saw a 10% reduction.

Table 4.4: Comparation of execution cost for traceability contract methods when using event-based and direct smart contract storage for URI

Method Name	Contract storage	Event-based storage	
`handleUpdate`	135,279	44,115	67% reduction
`handleTransaction`	172,659	86,395	50% reduction
`newBatch`	264,665	156,099	41% reduction

Figure 4.5[12] presents a comprehensive comparison of all the solutions discussed, featuring linear trendlines that fit the data with a R^2 value of 1. This indicates that there are no deviations from this equation, allowing for reliable experimental approximations of Gas cost for both on-chain data storage and the use of smart contract events. This results in Gas unit values of 946 and 37.5 per document byte,

[12] Please note that one series may appear to be missing, but it is actually hidden behind the red line due to their small difference.

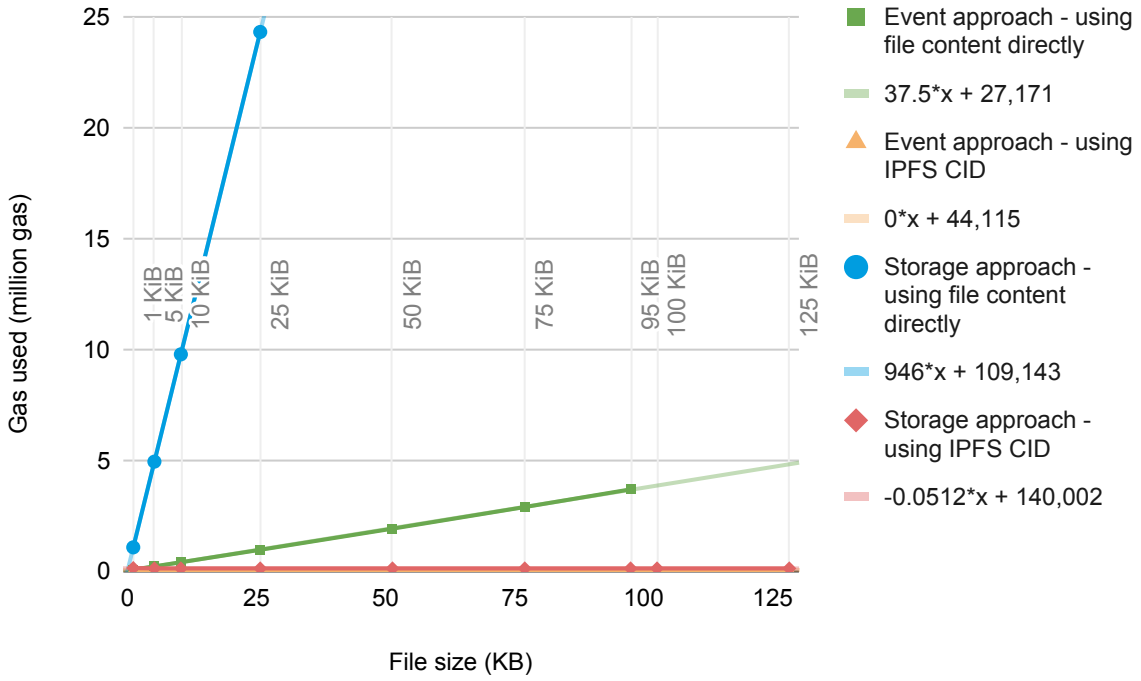


Figure 4.5: Comparison between approaches and their respective trendlines

respectively.

Notably, the trendline for the IPFS CID approach using events is in fact constant, as the CID length remains the same regardless of the file size, as previously mentioned. In contrast, when storing the CID directly on-chain, the trendline is nearly constant but not precisely as we suggested earlier. This is attributed to the need to initialize the storage variable, particularly the array length, during the first update of a batch in that approach. In the EVM, 20,000 Gas units are paid for an SSTORE operation when transitioning from zero to a non-zero storage value, compared to only 2,900 for subsequent storage store operations. This cost, however, is amortized over several updates and can be effectively considered negligible.

4.3.2 Workload Evaluation

In this section, we provide an in-depth evaluation of a workload scenario of our traceability system and present associated Gas costs. The workload simulation is designed to represent real-world usage scenarios, evaluating various system features. This simulation is used to help understand the distribution of costs across different parts of the system and assess the overall efficiency of our solution.

The workload we have defined simulates almost complete utilization of the system features. Note that the execution cost associated with this workload will be incurred by different actors within the system,

and can be found in Figure 4.6 while providing a visual representation of the cost distribution between different operations.

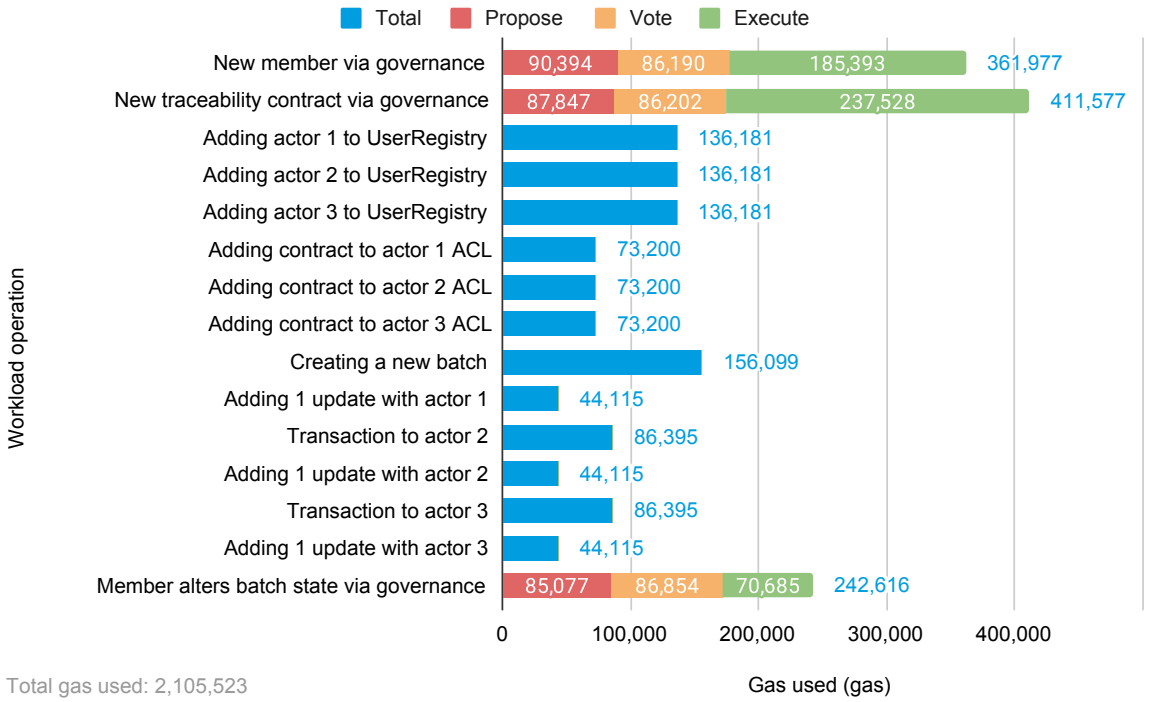


Figure 4.6: Distribution of Gas consumption for each workload operation

We define the workload by the sequence of the following operations:

1. **Registering a New Regional Member via Governance:** We start by registering a new regional member through the governance process. This process involves three phases: proposal, voting, and execution as depicted in Figure 4.6. The new member is assigned to oversee a specific traceability smart contract

2. **Creating a New Traceability Contract via Governance:** Next, we create a new traceability smart contract via governance. The new regional member proposes this action and becomes the contract manager.

3. **Adding Actors to the User Registry:** We add three new supply chain actors to the User Registry.

4. **Authorizing Actors and Adding Contracts to Capability List:** Each newly added actor is authorized by the regional member, and the recently created traceability contract is added to their capability list to enable interaction with the traceability system.

5. **Initializing a New Batch:** Actor number 1 initializes a new batch within the traceability system. This action will add the first transaction that will be part of our traceability information.

6. **Publishing Updates and Transactions:** Each actor publishes exactly one update and one transaction, simulating their engagement with the traceability system.

7. **Flagging a Batch with an Incorrect Conformity State:** The regional member concludes the workload by flagging the batch with an incorrect conformity state, representing a potential food safety issue.

It's important to emphasize that this evaluation includes a demonstration member with the authority, i.e. enough voting power, to independently accept proposals. In a real-world scenario, governance operations might require votes from multiple members until the proposal is approved, and each voting member will have to pay a Gas fee, multiplying the overall Gas consumption.

Indeed, our evaluation highlights some key findings. Governance operations are the most expensive. The governance process incurs the highest Gas consumption among all operations. Closer examination of Figure 4.6 in conjunction with the data in Table 4.3 reveals a substantial increase in Gas usage during the execution phase when compared to direct execution. On average, this increment amounts to approximately 32,000 Gas units. It is also worth reiterating a point previously discussed in Section 4.3: the proposal and vote phases consistently consume a similar amount of Gas regardless of the specific operation being executed. We anticipated the introduction of some overhead due to the presence of governance smart contracts, as this is common when maintaining a DAO, which is a central part of our system.

A second observation is that the setup phase and member management incur the most costs, but that the traceability portion of our system has been carefully optimized to reduce cost, as they are among the cheapest operations of our workload. This insight is critical, as we anticipate more interactions with the traceability contract than with governance operations in real-world use cases.

Lastly, we highlight that batch creation stands out as the most resource-intensive operation within the traceability interactions. This is primarily attributed to the necessary storage initialization process. Currently, this cost is borne by the actor who initiates the traceability system. However, it is clear that in the future, we should explore mechanisms to address this issue, such as potentially distributing the cost among all the actors involved in the process. This question is discussed further in Section 5.1.

4.4 Governance Improvements

In this section, we explore the enhancements and optimizations made to the OpenZeppelin governance smart contracts. OpenZeppelin is a big reference in the Ethereum ecosystem. It offers a suite of reference smart contracts crafted by a talented community and meticulously designed for both security and efficiency. Further optimizing these contracts, in their design, would be an arduous task with minimal gains. Instead, we aimed to adapt them to our specific context. OpenZeppelin provides contracts

that are designed to be abstract and versatile, allowing developers to make adjustments by modifying parameters rather than underlying implementations. This means that they have to support a degree of abstraction that translates into a cost overhead. Our approach was to fit their contracts, hopefully inheriting some of the security and efficiency, to our specific context removing this now unneeded abstraction layer and simplifying some of the features and design decisions that we deem unnecessary for our context.

The development of our system began by implementing the governance process as per the OpenZeppelin recommendations. After thorough testing to ensure compliance with our specific requirements, we arrived at the following contract definition:

```
1  contract GovernorContract is
2      Governor,
3      GovernorSettings,
4      GovernorCountingSimple,
5      GovernorVotes,
6      GovernorVotesQuorumFraction,
7      GovernorTimelockControl
8  {
9      constructor(
10         IVotes _token,
11         TimelockController _timelock,
12         uint256 _votingDelay,
13         uint256 _votingPeriod,
14         uint256 _quorumFraction
15     )
16         Governor("TracerDAO")
17         GovernorSettings(_votingDelay, _votingPeriod, 0)
18         GovernorVotes(_token)
19         GovernorVotesQuorumFraction(_quorumFraction)
20         GovernorTimelockControl(_timelock)
21     {}
```

This contract inherits several extensions, most notably “GovernorVotes” (line 18) which accepts an ERC-20 token contract address, and the “GovernorTimelockControl” (line 20) which receives the address of a timelock contract.

Our efforts to enhance the OpenZeppelin version of governance were directed at improving deployment and execution costs. The achieved values were notably better: The deployment cost of the

recommended implementation, which included all the governance-related contracts, initially stood at a substantial 8,335,718 Gas, equivalent to approximately 192.55 euros. Following our optimizations, the deployment cost was dramatically reduced to a mere 2,337,982 Gas, which translates to a significantly lower cost of 54.01 euros. This represents a 71% improvement in deployment cost.

Moreover, execution costs were also substantially enhanced, achieving a 50% reduction in Gas consumed by governance operations in our test suite. This is mainly due to removing the need for some method call entirely, a clear visual representation of which can be observed in Figure 4.7. Note that all values were obtained with compiler optimizations enabled and based on 1000 runs.

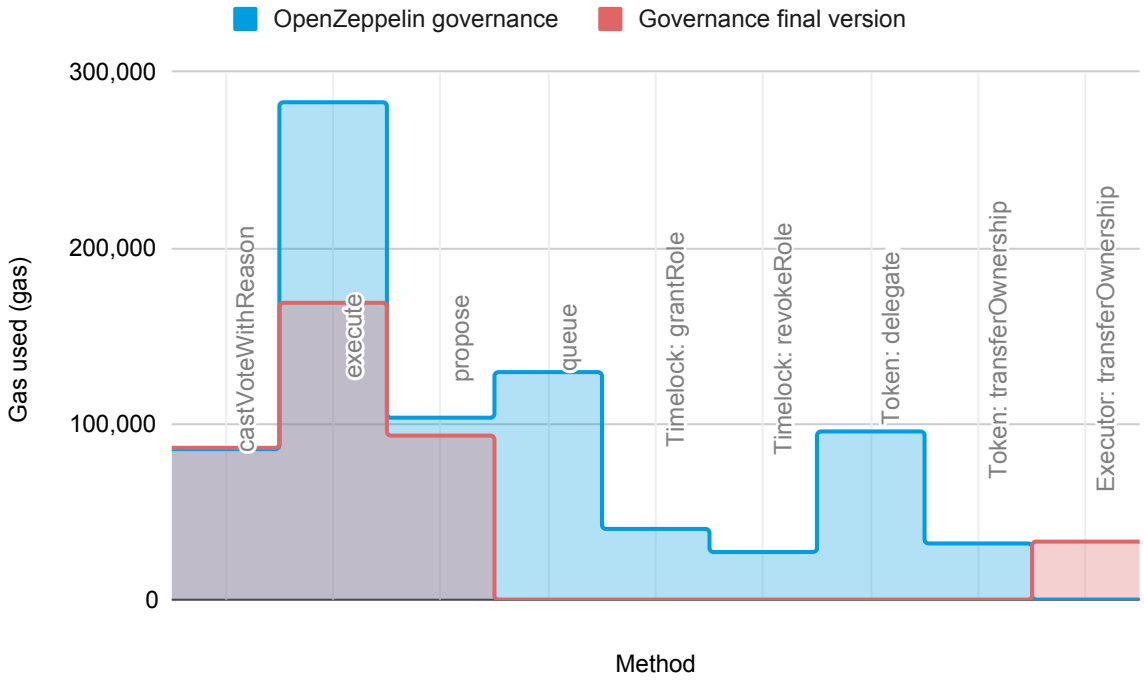


Figure 4.7: Gas consumption from a test set on both Governor implementations

To explain the achieved results we provide a walkthrough of the improvements performed:

Timelock removal: A timelock is a component typically used for introducing a waiting period. Generally, in a DAO it is good practice to add a timelock to governance decisions since it allows users who disagree with a decision to exit the system before it is executed. When employing a timelock, that specific contract will execute proposals, meaning it is designated as the transaction sender, and thus its address should be the one holding any funds, ownership, and access control roles. In our context, we made the strategic choice to entirely remove the timelock contract. This decision resulted in a noteworthy reduction of 1,697,947 Gas in deployment costs. The rationale behind this decision depends on the fact that the members of our system cannot choose to enroll or buy their entrance, instead their enroll-

ment has to go through a voting process. Once added to the system, they can choose not to participate in governance decisions but being part of the supply chain is inherently their function and if we want to mimic a global supply chain there is typically no exiting because a member does not agree with the others' decision.

Executor Contract: In the absence of the timelock, the responsibility for executing successful proposals shifts to the primary Governor contract. To align with best practices of maintaining separate contracts for distinct concerns, we later removed all executor capabilities from the Governor's contract and, instead, introduced a dedicated executor contract responsible for proposal execution, token management, and contract ownership. Although our DAO currently does not hold any ETH tokens, we have retained this feature within our executor contract to ensure future adaptability. Additionally, we have implemented measures against reentrancy attacks, which exploit vulnerabilities that arise when a malicious contract repeatedly calls another contract's function before the previous call concludes, potentially compromising the target contract's state and enabling unauthorized fund extraction by attackers.

Share-based membership: The recommended governance version relies on a Token-base membership, where ERC-20 tokens are used to authorize and quantify voting power, requiring the deployment and maintenance of a specialized ERC-20 token with an extension known as "ERC20Votes". This extension records historical balances, called snapshots, to prevent influence from users who acquire tokens solely for specific proposals and then sell them immediately after and most importantly, to prevent double voting. However, for our specific purpose, this approach introduces unnecessary complexity and overhead.

In response, we opted for an alternative approach of Share-based membership. We integrated the governance contract with our user registry to determine each member's voting power. This choice eliminates the need to deploy a costly token contract (2,397,804 Gas) and simplifies the process. With our "User Registry", members must be voted on and accepted by others, ensuring that voting power cannot be acquired at will, eliminating the need for snapshots of voting power, notably decreasing the overall execution cost. Moreover, we implemented maximum voting power limits that can be adjusted as needed, further enhancing flexibility and control.

By adopting this approach, we can also eliminate previously necessary features of the main governor contract like computing quorums using token snapshots, and support for receiving tokens ("IERC721-Receiver", "IERC1155Receiver") as we consider that, at least for now, holding tokens is not aligned with our governance model.

Removed support for extra parameters: Several methods that were rendered unnecessary by the recent modifications were removed. Additionally, we removed the ability to cast votes with additional parameters beyond the reason string. After aligning the contracts with our system's requirements, we found these methods to be unnecessary, as we have chosen to utilize an IPFS CID as the reason

string, allowing us to include any required additional parameters within IPFS. The previous approach involved emitting a distinct event to convey these additional parameters, and by eliminating support for such parameters, we not only reduced deployment costs by reducing the number of methods but also eliminated an entire event type.

Signature-based voting removal: We made the decision to remove support for casting votes with a signature, a feature that OpenZeppelin provides. This feature allows a user to vote on behalf of another user by using a pre-signed message, effectively delegating their vote. While this feature offers an intriguing possibility for the future, we opted to exclude it at present to minimize deployment costs, as its implementation also requires additional code to comply with EIP712 standards.

Front-running protections removal: We made the decision to remove certain front-running protections due to their diminished relevance in our context. A front-running attack involves a malicious actor observing pending transactions in the mempool and attempting to execute a similar transaction with a higher Gas price, effectively ”jumping in front” of the original transaction. This allows the attacker to gain an advantage, potentially manipulating the target contract's state or securing a favorable position in a trade or transaction. A typical example is arbitrating between an exchange bid. For example, an attacker observes a new transaction in the mempool, someone buying a big amount of ETH tokens. He can front-run his own transaction, buying ETH and selling right after the initially observed transaction finishes (after the ETH value has gone up), making a profit.

In the governance context, however, the value of this attack is less noticeable. An attacker could potentially steal a proposal in the mempool and claim credit for it. Furthermore, it is crucial that the executor's role is not solely assigned to the proposer, as this could enable an active attacker to disrupt the DAO's functioning, making all proposals their own and preventing any execution by others.

Previously, the contract had checks in place for valid proposal descriptions, avoiding front-running attacks by binding them to the proposal owner. However, in our context, where speed in decision-making is prioritized and the proposer does not have any special role, the need for these checks has diminished, and removing them resulted in reduced complexity and lower execution costs.

As a side note, it's worth mentioning that our traceability smart contracts are not susceptible to front-running attacks. This is because they restrict access to the current batch owner, meaning that only requests originating from the owner's account will be accepted.

Voting delay, threshold, and proposal states: We made several other simplifications. First, we removed the voting delay, allowing voting to commence immediately upon proposal submission. This change reflects our confidence in the behavior of all members, who are vetted and connected to real organizations, with real consequences for their actions. This approach ensures that if a proposal gains sufficient support and is deemed correct, it can be executed promptly, which is of extreme importance in scenarios like food hazard notifications.

The default implementation provides the option to set a proposal threshold, which determines the number of votes required for a voter to become a proposer. However, in our context where members must undergo a voting process to join, we chose to remove the threshold along with its smart contract logic. This change allows every member or actor in the registry to propose, regardless of their voting power.

Additionally, we eliminated the "pending" proposal state. Now, the voting process begins right after a proposal is submitted. However, we retained the ability for the proposal owner to cancel a proposal, but only before any votes have been cast. This adjustment is intended to prevent the unnecessary consumption of member resources.

Regarding the voting period, we chose to maintain it to prevent the acceptance of a dormant proposal. Once a member casts their vote, it cannot be undone, and the voting period ensures that the vote remains valid only for the specified duration.

In summary, this section aims to illustrate that while OpenZeppelin contracts serve as an excellent foundation, it is feasible, albeit not trivial, to further optimize these contracts by eliminating features that are unnecessary within the context of our application.

4.5 Gas Optimization

Efficient design decisions, such as the use of a clone contract factory, play a pivotal role in improving the cost-efficiency of our solution. While these design decisions affect Gas consumption the most, there are also some guidelines and patterns that can help optimize our smart contracts. In this section, we will discuss some that were taken into consideration while developing our system.

Several optimization techniques exist in Solidity. Aside from the more common use of immutable types and method scope (public), we highlight some more.

- Packing storage variables together: In the EVM, data is stored within 256-bit slots. To optimize storage efficiency, we can pack multiple smaller-type variables inside a single slot. The compiler automatically performs this packing, but it's important to note that variables must be declared consecutively to enable this optimization.

```
uint8 x     /* slot0 (8 bits) */              uint256 y   /* slot0 (256 bits) */
uint256 y   /* slot1 (256 bits) */    vs      uint8 x     /* slot1 (8 bits) */
unit8 z     /* slot2 (8 bits) */              unit8 z     /* slot1 (8 bits) */
```

- Using `calldata` instead of `memory`: In the EVM, there are three data locations: `storage` for permanent contract state, `memory` for temporary data during contract execution similar to stack memory,

and `calldata` for function input data. When defining function parameters with the `calldata` keyword instead of memory, it means the function reads the data directly from the transaction's input data payload. This approach is generally more Gas-efficient because it avoids the need to copy data into memory, which consumes additional Gas. However, it can only be used for read-only parameters. If modification or storage of data is required, a copy operation to memory is necessary.

- Caching storage variables: Caching storage variables can greatly reduce Gas costs, especially when frequently accessing the same storage variables in your smart contracts. Gas usage can be optimized by copying these values into memory, converting multiple SLOAD, i.e. EVM storage load operation, into just one, with subsequent MLOAD operations, i.e. EVM load memory data operation.

```
function checkAccess(...) public view returns (bool) {
    // caching the length
    uint len = actors[addr_].participatingContracts.length;
    for (uint i = 0; i < len; i++) {    // instead of "i < array.length"
        // ...
    }
}
```

- Choosing the correct data type (Mapping vs Array): In Solidity, developers have the option to use mappings or arrays as dynamic data types for storage variables. The choice depends on the specific use case. Mappings are ideal for direct access using known keys, resembling a hashmap behavior by computing the key's hash and accessing the data slot directly. Arrays, on the other hand, allow direct slot access via an index but require iteration for finding elements, incurring additional storage access costs. While mappings are generally more cost-effective, in some cases, iteration may be necessary for the application, and this feature is not available in mappings.

- Using custom errors: When handling errors in Solidity, developers have two options. One is to revert directly with a string message, while the other is to define a custom-named error and use it directly. The latter option is more cost-effective and allows for including additional arguments in the error message, making it convenient for providing dynamic error information.

```
require(condition, "Error message");
// vs
if (!condition) revert CustomError(additionalData);
```

5

Conslusion

Contents

Upcoming regulations and increased awareness among consumers, translate into an increasing need for traceability systems that satisfy most customer requirements. Blockchain technology is well suited for use in supply chain traceability due to its decentralized, immutable, and secure nature.

We propose a generalized supply chain traceability system supported by smart contracts in a public Ethereum network. The system can be used by a customer to reconstruct product movements throughout its lifecycle in the supply chain which allows consumers to make better-informed decisions.

In addition, the use of a DAO for regulatory oversight allows for decentralized decision-making while still providing a regulatory body to ensure that actors in the supply chain are acting in accordance with social and environmental norms. A novel approach that we have not yet seen explored in other traceability systems.

Furthermore, our system provides a set of user interfaces, most notably an interface targeted at consumers featuring an intuitive timeline design to enhance user recognition and interaction. To access this interface, consumers simply scan a QR code conveniently provided on the product packaging. The

QR codes within our system are generated in a way that can also be used to replace normal barcode functionality at a store checkout point.

One important consideration in blockchain development is the associated costs, both deployment and execution expenses, as suboptimal design decisions can easily translate into a more expensive system. Acutely aware of the importance of cost optimization, particularly given our system's potential for widespread usage across numerous products, we conducted a comprehensive analysis to identify potential areas for improvement.

We worked to refine various aspects of our system but the two major areas of focus were off-chain storage solutions and governance contract improvements.

Regarding data storage, we conducted an evaluation of various methods, ranging from direct on-chain storage to the use of off-chain content URIs. Eventually, we opted for a solution that leverages Ethereum events and document URI for efficient and cost-effective storage.

Furthermore, we implemented an access control mechanism for our IPFS system, tightly integrated with our on-chain DAO. This measure serves to prevent any unintended or malicious exploitation of our system resources.

Ultimately, our efforts to refine the conventional OpenZeppelin implementation of governance contracts yielded impressive results. We managed to achieve a 71% reduction in the deployment cost of our DAO.

In conclusion, we aspire to have demonstrated the necessity for a system like the one proposed, and we hope that our work can contribute to and enhance further developments in this domain.

5.1 Discussion

In our traceability system, each producer is assigned an address, a unique key with public key encryption characteristics, and that address is registered in the DAO along with additional information that identifies the actor. In the blockchain system, transactions originating from that producer are digitally signed with his private key, ensuring non-repudiation. This property, among others, inherited from the blockchain system, makes for a robust certification system for a company's own products, since any counterfeiting company will not have the private key that matches the public key of the producer so they cannot spoof their entity as a legitimate company.

Furthermore, as actors must undergo a vetting process prior to gaining transaction privileges within the traceability contract, the contract inherently serves as a certification authority. For instance, consider a member responsible for the Rainforest Alliance Cocoa certification. Within the associated smart contract, only certified actors are granted access, and all registered batches are inherently certified as well.

While this property is often overlooked in other systems, it provides a strong certification system with little adaptation from the blockchain platform, making it a key benefit of a blockchain traceability system.

A challenge that our traceability system, along with other proposed solutions, cannot mitigate is having a bad actor along the supply chain interfere with the physical contents of a packaged product our submitting fake information as updates. If it is evident enough for the receiving actor to detect there were any misdoings, he could request an investigation by the DAO (as explained in Section 3.1.3). While we provide no way to detect this with certainty, if any issues derive from that action it will injure the company since the product shows in the traceability system as originating from them. Therefore, companies are incentivized to ensure that the products in their batches remain unchanged, as any tampering would damage their reputation and potentially lead to negative consequences, so it would be in their interest to track down the culprit, and for that, they can leverage the traceability system in place since the misbehaving actor has necessarily recorded a transaction.

To address this challenge, companies like Everledger[1] seek to develop anti-tampering solutions, which, in this instance, are designed for the wine and spirits industry and are reliant on NFC technology.

It is also worth discussing our choice of blockchain platform. Ethereum is the most popular and well-established smart contract platform currently available, making it a natural option for our traceability system. It makes even more sense to adopt this technology in the initial stages of our system, compared to others such as Hyperledger Fabric, since it would require no initial investment in infrastructure.

Permissioned blockchain systems like Hyperledger Fabric may offer improved performance, but we believe that the maturity and wide adoption of Ethereum outweigh the potential benefits for now, especially because transaction speed is not a big concern for our system since there is usually some time until the product reaches the consumer.

Nevertheless, it may be worthwhile to explore the potential of leveraging our governance system for access control in a permissioned blockchain. This idea will be further elaborated in Section 5.2. Furthermore, one of the distinctive features of our system is its modularity. This modularity empowers us to transition to a different blockchain platform should the need arise, with minimal requirements for extensive system interface refactoring. By adjusting the underlying mechanisms of our API to support the new blockchain platform, the applications built on our system can remain relatively unaltered, offering enhanced flexibility and adaptability for the future.

Next, we will discuss some implementation-specific issues.

Initiating a batch in our traceability system comes with added costs that pose an important consideration. As seen in Section 4.3, creating a new batch costs almost twice as much as submitting a

[1] `https://everledger.io/anti-tamper-solutions/`

transaction. The burden of these costs, we believe, should be a matter of shared responsibility or even potential reimbursement to the original owner.

Furthermore, the question of who holds the responsibility for initiating a new batch arises. Should it rest with the vineyard owner or the wine processor? In certain instances, is the same entity handling both roles? Deciding on the precise inception point of our product lifecycle is another complex aspect. Does it commence with the grain or the bread? As consumers, our preference is to access the full journey, from the origin and processing but many companies only identify a batch when a product materializes.

As of now, we opt to leave this decision to the discretion of our system users, with the expectation that consumer demand for comprehensive traceability information will prompt companies to include as much relevant data as possible. A potential mitigation strategy involves retroactively including previous transactions with raw material providers as updates after batch initialization by a later actor. This approach necessitates careful implementation to align with our system requirements. One viable solution could involve the use of pre-signed transactions between actors at the time of the transaction, which could then be added as updates after batch initialization, authenticated by both parties through their associated Ethereum accounts.

Another concern relates to the risk of losing access to batches due to incorrect recipient addresses. Similar to the operation of Ethereum token transfers, if a wrong address is specified as the recipient, the batch could become inaccessible, effectively consigning the batch to an unusable address. To mitigate this issue we currently check if the provided receiver is an authorized actor for that traceability contract, drastically improving the risk of the batch becoming inaccessible.

Like any technological solution, there are advantages and trade-offs associated with our system. However, from a consumer perspective, we strongly believe in the potential benefits. This system empowers consumers to make more informed purchasing decisions and, notably, provides a crucial mechanism for identifying and addressing affected products in the event of a food hazard.

5.2 Future work

In this section, we identify aspects of our project that could be improved and possible ideas to pursue in the future.

1. Governance Interface: Regrettably, due to time constraints, we were unable to complete the implementation of the DAO management interface. While it is currently possible to interact directly with the smart contract, which we consider feature-complete, creating an intuitive user interface is necessary to facilitate governance operations.

2. Private or Hybrid Blockchain: Building upon our earlier discussion, the fact that our system already

integrates a governance process could be used as a stepping stone towards migrating to a permissioned blockchain controlled by the DAO. In this scenario, the same procedures employed for adding members and actors to our user base would also extend to authorizing their addresses within the access control mechanisms of the permissioned blockchain.

The use of a permissioned blockchain offers potential benefits in terms of performance, reduced latency, and especially, execution cost. We could either move the system fully over to a permissioned blockchain or implement a hybrid version where the traceability contracts would execute on the permissioned platform, since there is more frequent interaction with them making this interaction cheaper, and the governance part, our DAO, would be kept on public Ethereum, which could be considered more secure due to the number of available nodes. The potential interaction between distinct blockchain technologies, potentially facilitated through an Oracle contract, could be an interesting area for future research.

3. Resource Limitations and Controlled Product Exploration: Another valuable application of our system is its potential role in addressing the issue of overexploitation of natural resources. Specifically, it could play a pivotal role in scenarios where strict resource limits are mandated, such as the case of overfishing, where fishermen are permitted to capture only a restricted quantity of fish. Our system could adapt to enforce these restrictions effectively. The use of a single uniform system could facilitate checking for compliance with the imposed limitations. A possible approach could be the use of ERC-20 Tokens that represent and limit the total share of that restricted product.

4. Batch Identifier: In our system, we currently offer support for linking a Global Trade Item Number (GTIN) with the batch ID through the utilization of the GS1 Digital Link standard. This is necessary because we want to support the use of our generated QR code at the store checkout.

   However, it seems redundant to have a GTIN associated with the product because the batch identifier could unequivocally identify the product. This possibility opens the door to the development of a decentralized and transparent system that relies on cryptographic unique identifiers for this specific purpose. Such a system would eliminate the reliance on external organizations like GS1, which provide GTIN uniqueness as a paid service.

5. IoT Integration: As proven by other works in the field, the integration with the Internet of Things (IoT) could help mitigate user errors and enhance the overall efficiency of our traceability system. This integration can be achieved as an additional layer without altering the core functionality of the system.

Bibliography

[1] U. N. D. of Economic and S. A. P. Division, "World Population Prospects 2022: Summary of Results," *UN DESA/POP/2022/TR/NO. 3.*, 2022.

[2] F. De Canio and E. Martinelli, "EU quality label vs organic food products: A multigroup structural equation modeling to assess consumers' intention to buy in light of sustainable motives," *Food Research International*, vol. 139, p. 109846, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0963996920308711

[3] P. (Nel) Wognum, H. Bremmers, J. H. Trienekens, J. G. van der Vorst, and J. M. Bloemhof, "Systems for sustainability and transparency of food supply chains – Current status and challenges," *Advanced Engineering Informatics*, vol. 25, no. 1, pp. 65–76, 2011, rFID and sustainable value chains. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474034610000455

[4] A. Zhang, A. Mankad, and A. Ariyawardana, "Establishing confidence in food safety: is traceability a solution in consumers' eyes?" *Journal of Consumer Protection and Food Safety*, vol. 15, no. 2, pp. 99–107, 2020. [Online]. Available: https://doi.org/10.1007/s00003-020-01277-y

[5] R. Granillo-Macías and E. O.-B. Isidro J. González-Hernández, "Blockchain for Agri-Food Supply Chain Traceability," *International Conference on Industrial Engineering and Operations Management*, 2021.

[6] F. 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.

[7] E. Scallan, R. M. Hoekstra, F. J. Angulo, R. V. Tauxe, M.-A. Widdowson, S. L. Roy, J. L. Jones, and P. M. Griffin, "Foodborne illness acquired in the United States—major pathogens," *Emerg. Infect. Dis.*, vol. 17, no. 1, pp. 7–15, Jan. 2011.

[8] N. Patelli and M. Mandrioli, "Blockchain technology and traceability in the agrifood industry," *Journal of Food Science*, vol. 85, no. 11, pp. 3670–3678, 2020. [Online]. Available: https://ift.onlinelibrary.wiley.com/doi/abs/10.1111/1750-3841.15477

[9] M. P. Kramer, L. Bitsch, and J. Hanf, “Blockchain and Its Impacts on Agri-Food Supply Chain Network Management,” *Sustainability*, vol. 13, no. 4, 2021. [Online]. Available: https://www.mdpi.com/2071-1050/13/4/2168

[10] A. Srivastava and K. Dashora, “Application of blockchain technology for agrifood supply chain management: a systematic literature review on benefits and challenges,” *Benchmarking: An International Journal*, vol. 29, no. 10, pp. 3426–3442, Jan 2022. [Online]. Available: https://doi.org/10.1108/BIJ-08-2021-0495

[11] M. P. Caro, M. S. Ali, M. Vecchio, and R. Giaffreda, “Blockchain-based traceability in Agri-Food supply chain management: A practical implementation,” in *2018 IoT Vertical and Topical Summit on Agriculture - Tuscany (IOT Tuscany)*, 2018, pp. 1–4.

[12] R. L. Rana, C. Tricase, and L. De Cesare, “Blockchain technology for a sustainable agri-food supply chain,” *British Food Journal*, vol. 123, no. 11, pp. 3471–3485, Jan 2021. [Online]. Available: https://doi.org/10.1108/BFJ-09-2020-0832

[13] P. Ciaian, “Use of blockchain applications in the agri-food sector: state of play,” *Joint Research Centre, European Commission*, 2020.

[14] Council of EU, “Regulation (EC) No 178/2002 of the European Parliament and of the Council of 28 January 2002 laying down the general principles and requirements of food law, establishing the EFSA and laying down procedures in matters of food safety,” 2002, retrieved from https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32002R0178.

[15] UNDP, “Blockchain for Agri-Food Traceability,” *Singapore: UNDP Global Centre for Technology, Innovation and Sustainable Development*, 2021.

[16] K. Salah, N. Nizamuddin, R. Jayaraman, and M. Omar, “Blockchain-Based Soybean Traceability in Agricultural Supply Chain,” *IEEE Access*, vol. 7, pp. 73 295–73 305, 2019.

[17] L. Cocco, K. Mannaro, R. Tonelli, L. Mariani, M. B. Lodi, A. Melis, M. Simone, and A. Fanti, “A Blockchain-Based Traceability System in Agri-Food SME: Case Study of a Traditional Bakery,” *IEEE Access*, vol. 9, pp. 62 899–62 915, 2021.

[18] Fairtrade International, “Key benefits of Fairtrade.” [Online]. Available: www.fairtrade.net/about/key-benefits-of-fairtrade

[19] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[20] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger, Berlin version.” [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[21] D. Yaga, P. Mell, N. Roby, and K. Scarfone, “Blockchain Technology Overview (NISTIR-8202),” *NIST: National Institute of Standards and Technology*, 2018.

[22] A. M. Antonopoulos and G. Wood, *Mastering ethereum: building smart contracts and dapps*. O’reilly Media, 2018.

[23] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, “Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains,” in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys ’18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3190508.3190538

[24] M. E. Peck, “Blockchains: How they work and why they’ll change the world,” *IEEE Spectrum*, vol. 54, no. 10, pp. 26–35, 2017.

[25] S. Hassan and P. D. Filippi, “Decentralized Autonomous Organization,” *Internet Policy Review*, vol. 10, no. 2, pp. 1–10, 2021. [Online]. Available: http://hdl.handle.net/10419/235960

[26] M. I. Mehar, C. L. Shier, A. Giambattista, E. Gong, G. Fletcher, R. Sanayhie, H. M. Kim, and M. Laskowski, “Understanding a revolutionary and flawed grand experiment in blockchain: the DAO attack,” *Journal of Cases on Information Technology (JCIT)*, vol. 21, no. 1, pp. 19–32, 2019.

[27] A. Shahid, A. Almogren, N. Javaid, F. A. Al-Zahrani, M. Zuair, and M. Alam, “Blockchain-Based Agri-Food Supply Chain: A Complete Solution,” *IEEE Access*, vol. 8, pp. 69 230–69 243, 2020.

[28] G. Baralla, A. Pinna, R. Tonelli, M. Marchesi, and S. Ibba, “Ensuring transparency and traceability of food local products: A blockchain application to a Smart Tourism Region,” *Concurrency and Computation: Practice and Experience*, vol. 33, no. 1, p. e5857, 2021. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5857

[29] H. Feng, X. Wang, Y. Duan, J. Zhang, and X. Zhang, “Applying blockchain technology to improve agri-food traceability: A review of development methods, benefits and challenges,” *Journal of Cleaner Production*, vol. 260, p. 121031, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0959652620310787

[30] Z. Zhai, J. F. Martínez, V. Beltran, and N. L. Martínez, “Decision support systems for agriculture 4.0: Survey and challenges,” *Computers and Electronics in Agriculture*, vol. 170, p. 105256, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169919316497

[31] A. Kamišalić, R. Kramberger, and I. Fister, “Synergy of Blockchain Technology and Data Mining Techniques for Anomaly Detection,” *Applied Sciences*, vol. 11, no. 17, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/17/7987

[32] H. Chen, Z. Chen, F. Lin, and P. Zhuang, “Effective Management for Blockchain-Based Agri-Food Supply Chains Using Deep Reinforcement Learning,” *IEEE Access*, vol. 9, pp. 36 008–36 018, 2021.

[33] A. Tharatipyakul and S. Pongnumkul, “User Interface of Blockchain-Based Agri-Food Traceability Applications: A Review,” *IEEE Access*, vol. 9, pp. 82 909–82 929, 2021.

[34] L. Marchesi, M. Marchesi, G. Destefanis, G. Barabino, and D. Tigano, “Design Patterns for Gas Optimization in Ethereum,” in *2020 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, 2020, pp. 9–15.

[35] J. Benet, “IPFS - Content Addressed, Versioned, P2P File System,” 2014. [Online]. Available: https://github.com/ipfs/papers/raw/master/ipfs-cap2pfs/ipfs-p2p-file-system.pdf

[36] “Filecoin Spec.” [Online]. Available: https://spec.filecoin.io/

[37] Swarm, “Storage and Communication Infrastructure for a Self-Sovereign Digital Society.” [Online]. Available: https://www.ethswarm.org/swarm-whitepaper.pdf

[38] M. Steichen, B. Fiz, R. Norvill, W. Shbair, and R. State, “Blockchain-Based, Decentralized Access Control for IPFS,” in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2018, pp. 1499–1506.

[39] GS1, “GS1 Digital Link Standard: URI Syntax,” 2023. [Online]. Available: https://ref.gs1.org/standards/digital-link/uri-syntax/