MAASTRICHT UNIVERSITY

MASTER'S THESIS

User Incentives on an Ethereum Blockchain Data-Sharing Platform

Author:
Leonard PERNICE
Stud. Num.: 6041835

Supervisor:
Assoc. Prof. Visara UROVI

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

January 19, 2021

Declaration of Authorship

I, Leonard PERNICE, declare that this thesis titled, "User Incentives on an Ethereum Blockchain Data-Sharing Platform" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

MAASTRICHT UNIVERSITY

Abstract

School of Business and Economics

Master of Science

User Incentives on an Ethereum Blockchain Data-Sharing Platform

by Leonard PERNICE

In this thesis we explore how a range of incentives may be applied to a blockchain-based data-sharing platform. We identify challenges inherent to data-sharing and provide solutions to each by extending the LUCE data-sharing framework conceptualized by Havelange et al., 2019. We adapt the smart contract logic to include payment options, implement modules to increase general feasibility in terms of security, data ownership, and control mechanisms, and finally simulate how occurring costs from using the system may be covered. With this, we provide a baseline comparison for future iterations of the LUCE platform, as well as the basis for tests in closed real-world environments. The evaluation of our simulation indicates that accruing costs are small compared to current data-sharing costs, and that they can be covered efficiently.

Keywords blockchain, smart contract, data-sharing, EHR, incentives

Acknowledgements

I would like to express my gratitude to my supervisor Visara Urovi for the useful comments, remarks and engagement through the learning process of this master thesis. Furthermore I would like to thank Vikas Jaiman for his continued guidance and support on the way. I would like to thank my loved ones, who have supported me throughout entire process, both by keeping me harmonious and helping me putting pieces together.

Contents

List of Figures

List of Tables

List of Abbreviations

PHI	**P**rotected **H**ealth **I**nformation
ETL	**e**xtraction, **t**ransformation, **l**oading
GDPR	**G**eneral **D**ata **P**rotection **R**egulation
EHR	**E**lectronic **H**ealth **R**ecords
LUCE	**L**icense acco**U**ntability and **C**omplianc**E**
TAM	**T**echnology **A**acceptance **M**odel
PoW	**P**roof **o**f **W**ork
PoS	**P**roof **o**f **S**take
PoA	**P**roof **o**f **A**uthority
EVM	**E**thereum **V**irtual **M**achine
ERC	**E**thereum **R**equest for **C**omments
EIP	**E**thereum **I**mprovement **P**roposal
ADA-M	**A**utomatable **D**iscovery and **A**ccess **M**atrix
DUO	**D**ata **U**se **O**ntology

List of Symbols

μ	arithmetic mean
σ	standard deviation
$\mathcal{N}$	normal distribution
i.i.d.X	independent and identically distributed random variables

Dedicated to all those who hold out hope for science to find a cure to the disease that afflicts them.

Chapter 1

Introduction

Easy access to data is one of the main avenues to accelerate scientific research. Medical institutions are a source of an abundance of health-related data, but due to a variety of reasons, including privacy concerns, this data is not generally accessible to the scientific research community. If medical scientists had access to sufficient data, medical research and innovation would logically experience significant growth, which would in turn improve health care.

However, due to stringent laws protecting patients' health data, it is currently difficult, expensive, or even impossible to procure the large amounts of data that currently lies locked in the databases of hospitals and research institutions around the globe. At a minimum, sharing medical details pertaining to one or more patients (Protected Health Information, PHI) requires involving individuals in the data-sharing process. This process includes the expression of individual, active, and informed consent. Currently, patient consent is collected via consent forms that are created specifically for each study. Secondary use of data is typically possible if the original consent and the secondary use match. Not only is this process highly inefficient but often infeasible, and results in data being siloed instead of being used for research. The solution is third party involvement. Representative organizations collect data from consenting patients, or secondarily, from hospitals. Often, these organizations also provide the processes known as extraction, transformation, and loading (ETL) of the data, meaning they arrange the data in such a way that simplifies analytics. Finally, this data, collected and cleaned by such organizations, are then sold to those who can afford to pay the price. An example of this is the deal the direct-to-consumer genetics company 23andMe made with the biopharmaceutical giant

Genentech, to sell access to genomic data on roughly 3000 Parkinson's patients for a total of 60 million dollars (Molteni, 2018).

There is currently still no general, clear definition as to data ownership. The origin of the data, in this case the patient, is only one part of the equation. Other parties involved, such as doctors and hospitals also have legitimate ownership claims, and when the data is shared, it must first be processed and brought into a useful format, which involves the use of technology and human expertise that may turn out quite costly (Bourke and Bourke, 2020). This leads to fragmented ownership with many parties having a partial stake in the data, and thus inhibited economic growth (Duch-Brown, Martens, and Mueller-Langer, 2017). Regulators in Europe have made increasing efforts to provide the necessary legal basis for individuals to retain (or regain) control over their personal information, including PHI. Due to the General Data Protection Regulation (GDPR, European Union, 2016), individuals can request information on what data companies hold on them, why they do so, and who they share it with, as well as request rectification or deletion of data that pertains to them. The GDPR distinguishes between two types of data, personal and non-personal, and its regulations apply mainly to personal data. PHI falls into the category of personal data.

Following up on the example of 23andMe, the patients, who are the origin of PHI, are rarely adequately compensated for the data they provide, while those who sell the data enjoy enormous profits. It can be argued that this profit generated is at least partially justified since companies such as 23andMe make substantial investments into data-processing, which entitles them to compensation. Patients usually consent to their data being shared with third parties by agreeing to the license agreement of the respective service, without necessarily being aware of the connected potential risks and consequences. In the case of 23andMe, patients who share their data do so implicitly, unless they actively withdraw consent. This is a part of the service they purchase, which provides genomic ancestry information and possibly health-related findings that may or may not imply the risk of being affected by certain health conditions (23andMe.com, 2020). Next to these direct compensations, patients may indirectly benefit from the development of new medicines or treatments, resulting from biomedical research powered by the health data shared. Patients, however, typically will not receive monetary compensation, even if their data is sold at high prices.

This thesis makes advancements on an online data-sharing platform conceptualized by Havelange et al., 2019 that explores an alternative avenue to third-party data vendors. Its purpose is to empower data providers (e.g. hospitals) with the tools and incentives to actively control and share their health information with a broader biomedical research community. Several design features make this platform unique. First, it is a platform that relies on the publication of data[1], meaning it is completely dependent on the participation of health institutions such as hospitals, and other medical facilities. Publication, in this context, means the existence of the dataset will be visible, while its contents will be encrypted and accessible only for those with authorization. Since the data will be derived from general hospital records, the applications of the data on this platform will be very diverse. This makes it attractive to many different types of research. Secondly, the platform will allow data subjects to actively or even passively verify for what purpose, by whom, and in what timeframe their data is used.

The only realistic way to allow individuals intricate control of their owned data is an automated system. This can only function digitally, which makes it susceptible to digital attacks. This issue is compounded by the nature of the underlying system as a data-sharing platform. Data is quickly becoming the most valuable commodity in the world, as showcased in the previous example of 23andMe's deal with Genentech. This makes it a highly profitable target for cyber-attacks, which have already happened in the case of several electronic health records (EHR) projects (Ronquillo et al., 2018).

One solution to this problem might be blockchain-based encryption. Of course, blockchains introduce new problems, such as high maintenance costs, and sometimes limited scalability of blockchain applications (Digiconomist, 2020 and Wu, 2019). These issues will be addressed at a later point in this thesis (see chapter 7). In general, encrypting data through a public blockchain will provide the best possible security for the system (Gervais et al., 2016), since it is virtually impossible to hack (cipher.com, 2019), while still allowing those with the correct credentials easy

[1]Only the metadata pertaining to the content of the data is published. The data itself remains locked until access is granted by the system.

access from anywhere, given an internet connection. Thus, the data-sharing process could be automated using a licensing system that allows certain users to access the relevant data, possibly in return for some form of compensation.

Importantly, any transaction on a blockchain, such as granting access to certain data, or even registering a new dataset with the system or updating it, will incur a computational cost. In the Ethereum network, this cost is denoted by a *gas* value (Rosic, 2018). On the platform we showcase and extend in this thesis, the cost of updating a dataset scales with the number of users accessing each dataset, as any update must, by law (GDPR, Article 16), perpetuate through the system and thus ensure any copies of the relevant dataset are equally updated. Any *gas* cost resulting from a transaction will be incurred by the party that issues the transaction. Thus, if a data provider issues an update to their dataset, they will bear the inherent cost. This can quickly become prohibitive if the respective dataset is shared a large number of times. This risk may discourage potential data providers from using the system since they stand to incur costs. As such, the system proposed in this thesis implements necessary measures to control and track to whom costs apply and who covers them.

In particular, it stands to reason that any data requester should be held accountable to provide reasonable compensation in return for access to the data they need or at least take part in covering the cost accumulated by providing said data. The willingness of pharmaceutical and similar companies and corporations to pay large sums of money in return for data has already been established. However, academic researchers will likely lack the ability to procure comparable sums for the data they need (MRC, 2020).

This thesis thus proposes, explores, and implements a system that provides data to academic scientists at an affordable, reasonable price, and might have a positive impact on the pace of innovation as a whole.

The architecture and conceptual basis of the platform we realize in this thesis were introduced by Havelange et al., 2019 as the License accoUntability and CompliancE (LUCE) framework for data-sharing. Havelange et al., 2019 describe LUCE as a blockchain application that allows its users to share and reuse data in compliance with license agreements. Angerer, 2019a extend their work by establishing and implementing a working, reproducible prototype to support and simplify further research. Their prototype implements the general architecture conceptualized by Havelange et al., 2019 and provides all basic features necessary to demonstrate functionality. However, several key functionalities in their smart contract, such as registration and access requirements were oversimplified. Another trivialized feature in the prototype of Angerer, 2019a is the consent mechanism for data access requests. Jaiman and Urovi, 2020 are developing a dynamic consent control feature to add to the LUCE platform in parallel to this thesis, which will also be applied in the specific environment of EHR systems. With their consent mechanisms, individuals can precisely control the conditions under which their data can be accessed.

Tenopir et al., 2011 examine incentives that motivate data-sharing, and establish several key benefits that arise from it (verification of previous research, new interpretations, improved data integrity, resource optimization, guard against falsification, facilitation of researcher training). However, they also point out the lack of existing data-sharing infrastructure to safely and sustainably facilitate the practical aspects of sharing data. Similarly, Lo and DeMets, 2016 point out the benefits arising from data-sharing, specifically in clinical trials. As one possible incentive, they mention the provision of the National Institutes of Health that endorses data-sharing in grant reviews, which affiliates academic rewards as compensation for peer-to-peer data-sharing.

Since there is not yet any expansive data-sharing infrastructure in place, it is necessary to create a platform that facilitates incentive structures to motivate data-sharing on a larger scale. In light of this, the research question we seek to answer in this thesis is:

What incentives will motivate users to interact with the LUCE platform and how can these be realized and supported through technology?

Thus, the purpose of this thesis is to extend the LUCE platform by, (i) exploring incentive mechanisms to motivate user participation, (ii) creating a functioning prototype from the resulting incentive model, and (iii) analyzing the solution in scenarios simulating real-life user interactions. Specifically, the scenarios will showcase the accruing operational costs inherent to the blockchain implementation, and cross-reference those with the incentive formats and conceptual ideas logically linked to user growth, as formulated in this thesis.

By conducting these simulations based on a data-sharing platform that incorporates user incentives, we will create a foundation that provides and showcases the boundary conditions of a blockchain-based solution for data-sharing.

Our approach contributes a novel, but vital viewpoint on the platform, and it takes a significant step towards the practical realization of the project. It also shows what costs can be expected from a sophisticated data-sharing system based on blockchain technology, where they apply, and how they can be covered, as well as details on how the internal systems of smart contract logic can contribute to the implementation of GDPR compliance.

The remainder of the thesis is organized as follows: In chapter 2 we will explain the background of the LUCE platform and compare it to other, existing platforms to delineate its classification in the data-sharing sector. Chapter 3 serves to familiarize with the architecture of our extension to the LUCE platform and to propose in which system modules user incentives might be most effectively implemented. Chapter 4 clarifies what extensions we make to the LUCE prototype, as well as the methodology employed to achieve this, while chapter 5 explains how we simulate system behavior in concurrence with different incentives to the user. Chapter 6 will visualize the results of our simulations, and how the advancements made to the LUCE prototype mesh with the requirements for the platform in a practical, artificial environment. Finally, in chapters 7 and 8, we will discuss limitations of the methods and concepts used in this thesis, of the platform itself, and of the underlying technology, then suggest what future research may alleviate or solve these problems, and finally offer our conclusion.

Chapter 2

Background

Like many other institutions, hospitals and similar medical service providers have begun venturing into the digitalization of their data to improve the efficiency and simplicity of their processes. Yeung, 2019 found that the implementation of electronic health records improved both health of patients and life-expectancy. However, not all aspects of EHRs are immediately, strictly positive. The transition from paper-based documentation to electronic health records has been shown to increase documentation time during the adaption period, which points towards an initial reluctance in terms of acceptance of the new technology (Walker et al., 2019). Thus, it is reasonable to assume that newer systems such as blockchain applications will run into similar barriers to acceptance. The technology acceptance model (TAM) by Davis, Bagozzi, and Warshaw, 1989 and its continuation (Davis and Venkatesh, 2000) shed light on how new technology is received and adopted inside an organization. According to the results presented by Davis and Venkatesh, 2000, the most important factors in the acceptance of new technology is the subjective norm (what workers in that field view as standard compared to what the system is capable of) and perceived usefulness of the system. In EHR systems, both of these are closely related to the ease of use. However, such a system alone is of little immediate benefit to patients, other than the anticipated improvement to general healthcare that is predicted after the implementation of an EHR system.

This is why we propose a complementary system that works in concert with EHRs as a data distribution platform to establish the connection between medical researchers and data that is relevant to their studies.

The following sections explain the relevant concepts and systems that form the basis of this project. We aim to establish a link between EHR systems and researchers by combining the digital records of EHR with a data-sharing platform that is based on blockchain technology. This platform will allow researchers to inexpensively acquire relevant, and rich data to facilitate and expedite their respective research.

2.1 Data-sharing

The practice of data-sharing in everyday life has become so natural it is barely noticeable anymore. Platforms such as Facebook, Instagram, and others have made it extremely easy to share data, to the point where many are barely aware of it happening, and even less aware of the consequences of sharing this data. Targeted advertisement has become almost entirely unavoidable (Johnson, 2013). Default settings for these data-sharing sites specifically allow and support targeted advertisement, and changing these settings can sometimes be challenging or result in being denied access to the service. Theoretically, the benefit of targeted marketing is to provide the end-user precisely what they desire, when they desire it, and at an affordable price (Grier and Kumanyika, 2010). This systematic approach of analyzing people's behavior and then offering arguably useful products based on the results serve to massively stimulate the market, which leads to huge economic growth.

While the customers' benefit from targeted marketing is, ideally, getting exactly what they want or need, those who offer targeted marketing services also benefit from the customers' data, since these services are not offered for free. One of the problems herein lies with the balance of gain. One could argue that, in an ideal scenario, the benefit of allocating products to the exact people who want or need them far outweighs the revenues of those who analyze and sell the customers' data, which leads to that very allocation. However, the assumption of the ideal scenario is unrealistic, which means there are several cases where the allocation algorithm fails. In these cases, Grier and Kumanyika, 2010 argue the benefit to the customer may be zero, or even negative. This introduces an imbalance between the benefit of customers versus the benefit of those who implement the targeted marketing systems.

EHR systems are not specifically designed as data-sharing platforms. Instead, the purpose of EHRs is to provide a digital environment that largely eliminates the necessity of paper-based health records, and thus improve the efficiency of data input, review, storage, and ideally, security. As mentioned before, the implementation of an EHR system does not always and not immediately result in increased efficiency, and has even been shown to have adverse effects on efficiency after initial adoption (Marani et al., 2019). This may in some cases be due to unfamiliarity with the new system, or incomplete or lacking implementations, but can rarely be ascribed solely to technological issues (McGowan, Cusack, and Poon, 2008).

Data-sharing in the health sector carries different implications than targeted marketing. Organizations such as 23andMe, which focus on collecting medical data and sharing that data with third parties, also provide immediate services to their customers (data subjects) that go beyond what conventional healthcare providers generally offer (such as information of being at risk for certain medical conditions based on the individual's genetic code, 23andMe.com, 2020). That is not all these companies are offering. By sharing data with interested third-party research institutions and companies, they enable and drive medical research. The resulting medical innovation may in turn improve human health. Even if treatments derived from such research are almost always patented and sold at a substantial profit, they may in turn also offer substantial individual and public health benefits, as a direct, if presumably long-term consequence of biomedical data-sharing. This is an indirect benefit individuals might gain, depending on the availability, and price attached to the respective treatments.

Next to companies collecting and sharing data, another party in the position to share data are academic researchers. There has been research exploring direct, peer-to-peer data-sharing among researchers, which found the only feasible way to achieve such large-scale data-sharing (who would share their self-collected data) is a reputation payment system (Fecher et al., 2015). This would mean a data provider (here, a researcher that collected and/or has legitimate distribution rights over a particular dataset) would receive some form of credit whenever their data is used for further research. He et al., 2018 developed a distributed peer-to-peer blockchain based solution that facilitates and incentivizes data-sharing among peers (researchers).

From the observations above we can derive enablers and barriers to data-sharing, from which we will later extract data-sharing challenges.

All sharing of personal data is bound by the GDPR. Thus, the GDPR effectually can represent a barrier to data-sharing, which means any data-sharing platform must incorporate measures to ensure GDPR compliance of all parties involved. However, we can also see that GDPR compliance is not always enough. Often, services that many might see as indispensable parts of everyday life in this day and age, require data-sharing rights in return for the service. An example of this is Google, which collects and shares data from its users with their consent. Meanwhile, the data subject very rarely has any control over whom the data is shared with, which can lead to negative effects from data-sharing. At the same time, generally, data-sharing can be viewed as having a positive effect on economic growth, which should arguably result in an improvement of welfare in the great majority of cases. Thus, since data has such a high value and sharing it usually leads to positive outcomes (as long as privacy considerations are upheld), the basic act of data-sharing can be seen as an enabler, because its inherent value directly incentivizes sharing it. The problem that lies therein is the aspect of compensation. There are very few cases in which data subjects receive direct compensation in return for their data that is not directly represented by the respective service they are using (e.g. google.com). In the case of researchers and peer-to-peer data-sharing, there is also rarely any compensation in place.

From these perspectives, we can deduce three distinct challenges in data-sharing:

Challenge C 1 *Securing and enforcing data-ownership rights.*

Challenge C 2 *Enforcing access exclusivity, i.e. only particular groups can access the data.*

Challenge C 3 *Compensation in return for sharing the data.*

These challenges represent the boundary conditions of data-sharing. If the respective platform fails to provide solutions for these challenges, it will inevitably fail.

The LUCE platform aims to provide solutions for these challenges. We secure both ownership rights and access exclusivity by using blockchain technology and creating smart contracts (see section 2.2.2) that facilitate the precise nature of how and to whom data is shared, while implementing legally binding measures to ensure GDPR compliance of all users. Thus, the formed contract between the two parties, data provider and data requester, is enforced by a supervisory authority, which can passively observe the data-sharing process and to actively react to complaints. Similar to the idea of a reputation economy as explained by Fecher et al., 2015, we support and incentivize reputation gains by keeping a record of data transformation when the shared dataset includes records from multiple subjects, which is attributed to the originator. Additionally, since data shared on our platform usually originates from EHR databases, we effectively circumvent the financial interests of third-party commercial data vendors and promote research in all medical fields because both the search for data and its procurement is made very easy.

This data-sharing platform will work in close relation to EHR systems and offer similar indirect benefits as other biomedical data-sharing organizations, such as 23andMe. The key differences between our platform and others are its massive spread of data content accessible, high inherent security, the intricate individual control over personal data by data providers who represent data subjects, the possibility of direct monetary compensation in return for data, and the relative inexpensiveness of data being shared for those with the correct credentials (proposed as academic researchers instead of pharmaceutical and other corporations), as well as associated reputation gains by using the platform and possessing a specific license that allows access.

In the next sections, we will give an overview of blockchain technology and how it supports the goals of our platform in providing solutions to the challenges established above.

2.2 Blockchain

Blockchain technology is a cryptographic tool to facilitate transactions securely. It is a ledger that consists of transaction blocks and grows each specified time interval (e.g. every 5 minutes). As a ledger, its most common application is cryptocurrency. Each block contains the hash of the previous block, as well as a timestamp, and transaction data. The ledger is openly accessible and shared openly, worldwide, but transactions, once made and confirmed, are permanent and immutable (chriseth, 2020b). It is hosted on a large decentralized network, each node of which must accept a transaction before it becomes part of the blockchain. One of the mechanisms that protect against attacks on the system is a consensus mechanism called Proof of Work (PoW), a method to validate the next block.

Nakamoto, 2008, the inventor of Bitcoin, establishes that Proof of Work is a representation of which node in the network has put in the greatest effort, i.e. solved the most blocks, which represent the longest chain. To add the next block to the ledger, a *miner* must solve a complex mathematical problem, which is then verified by the node network. This means if the majority of nodes agree on *a* truth, then their chain will represent *the* truth. Each time this majority solves a new block, they will add to their chain. Since each new block is based on the previous block, an attacker would need to redo the current block and all blocks coming after it, and then catch up and surpass the longest chain. The probability of this succeeding diminishes exponentially with each newly added block to the longest chain.

Blockchains such as Bitcoin or Ethereum currently use proof of work, which means they are protected by computation. However, different consensus mechanisms do exist, one of these being Proof of Stake (PoS). The goal of Proof of Stake is to secure consensus without mining, which reduces (electricity) costs significantly[1]. PoS chooses the creator of the next block randomly based on their wealth (i.e. stake). The higher their stake, the higher the probability of being chosen to validate the next block and add it to the blockchain if it was verified by the network.

Regardless of the consensus mechanism, any transaction made is automatically cryptographically signed by the individual issuing it. This adds another layer of protection against specific modifications of the transactional data, meaning a simple check ensures a transaction extracting value from a certain wallet can only be issued while holding the corresponding private key (see below). Through simple functions that form the core of the consensus mechanisms of the Ethereum blockchain (and others), anyone can derive the canonical truth. This is why blockchains are often referred to as *trustless* environments.

[1]Since PoW requires great computational power to add new blocks to the blockchain, electricity costs of miners are generally quite high, which is why they are rewarded with cryptocurrency for successfully solving a block.

A private key is an authentication tool used to cryptographically verify the validity of a digital signature. Every private key is paired with a public key. The public key can be mathematically derived from the private key, but not vice versa[2]. The public key can be used to verify that a transaction was indeed issued by the holder of the private key. Thus, it is extremely important to always keep the private key secret and protected. The possession of the private key is equivalent to being its owner since there are no other mechanisms in place to prove otherwise. The private key is the only verification needed to access all funds associated with its corresponding public address (Massessi, 2018). This is why it is common practice to save the private key on a protected USB-stick or a hardware wallet specifically designed for this purpose.

However, there are several problems blockchain applications face. Transactions in a PoW-based network are inherently slow; a single transaction can take up to five minutes. This, among other problems, impedes the scalability of blockchain technology (Hazari, 2019). The cost of running a PoW-based blockchain can also be problematic (Digiconomist, 2020). Each transaction is associated with a large requirement for computing power, which is costly at a certain scale. Due to the costly nature of the technology, it is unavoidable that some form of incentive must be provided. This is why the so-called 'miners', who use their computing power to facilitate transactions, are rewarded for blocks they successfully solve and that are confirmed by the network.

Another problem that blockchains that implement cryptocurrency commonly experience is the volatility of the value attached to that currency. There is no centralized regulatory institution that controls the value of the underlying currency as is the case with conventional currencies (e.g. central banks). The value of cryptocurrencies is derived from its demand, meaning currencies in high demand will experience explosive growth in value, while those in low demand will quickly lose most value attached. However, some blockchains offer methods to circumvent this price fluctuation by establishing representative tokens. This will be exemplified in our section on tokens (section 2.2.3).

[2]The mathematical principle behind this is a trapdoor function, which easily computes in one direction, but requires exponentially escalating computing power in the other direction.

There are multiple benefits to using blockchain technology for our platform. Decentralization of public blockchains and their consensus mechanisms (PoW or PoS) provide excellent security for transactions. On our platform, these transactions will mainly consist of access rights to datasets being granted to researchers. Another benefit is the transparency offered by public blockchains, which enables any individual to keep track of transactions made, while anonymization protects individual privacy. Importantly, blockchain technology circumvents the need for third-party involvement, which will most likely result in a lower cost overall, but also eliminates the necessity for trust between users. Blockchain technology provides a trustless environment, as it is inherently impossible to abuse the system (under certain conditions, see section 2.2.2).

Apart from the public blockchains discussed above, other blockchain implementations offer similar tools, called permissioned blockchains. The most notable of these is the Hyperledger project. The key difference between Hyperledger and public blockchains lies in their respective architectures (Valenta and Sandner, 2018 and Sajana, Sindhu, and Sethumadhavan, 2017). Public blockchains are completely transparent and can be accessed to make transactions without any authorization, meaning they operate in a permissionless mode. The Hyperledger projects and similar blockchains are permissioned, meaning they can only be accessed by those who receive permission to do so by a usually centralized control structure. These two modes of operation each represent their own sets of advantages and disadvantages (Singh, 2020). The most prominent disadvantage of permissioned blockchains versus their public counterparts is that they are never truly decentralized. This means permissioned blockchains usually do not benefit from the trustless environment in public blockchains, because there is always some form of centralized control with theoretically limitless options to control or even manipulate transactions. This is represented clearly in the different consensus mechanisms employed in permissioned blockchains, such as Proof of Authority, which we will explain in more detail in section 2.4.2. As such, permissioned blockchains, similar to private ones, usually defeat the purpose of the technology, which is to provide a trustless, immutable environment where anybody can make transactions.

The advantage of permissioned blockchains is that they implement control over who can access the system, as required by our data-sharing challenge C2. However, with this thesis, we show that a similar level of access control can be achieved in public blockchains as well, which effectively preserves the trustless environment and still provides the prerequisites for data-sharing. Furthermore, the data-sharing challenge C1 dictates that control over shared datasets remains with the data providers / data subjects. This is why we conclude that permissioned blockchains are not suited to the goal of the LUCE platform. We base our application on a public blockchain called Ethereum, which will be explained in the next section.

2.2.1 Ethereum

Ethereum, developed by Buterin, 2013, provides a decentralized virtual machine (EVM) that is capable of executing scripts and uses Ether to pay for units of computation called *gas*. These scripts are the reason why Ethereum has great potential as the basis of a blockchain application that controls the process of data-sharing. It allows the creation of custom records on an immutable, decentralized ledger. However, as mentioned before this capability comes at a cost. The computational cost paid in *gas*, henceforth known as *transaction cost*, is attributed an Ether value, measured in Wei[3]. Ether is also used for other state transitions, such as simple value transfers from one wallet to another. Due to the existence of smart contracts (see section 2.2.2), Ethereum is highly flexible and offers an enormous range of possibilities regarding applications that require agreements between two parties. Whenever such an agreement is reached, i.e. consensus, the resulting transaction is written to the blockchain, which incurs the respective transaction cost. At this point it is important to note that there are types of functions in smart contracts that do not result in any *gas* cost. These functions are known as `view` functions and do not result in state transitions. This means the information stored on the blockchain remains unchanged and no transaction is written to the blockchain. Thus, it is inexpensive to read from the blockchain, but costly to write on it.

The reason we chose to run our proposed platform on the Ethereum network lies with its inherent advantages and nature, as well as its capability to execute custom scripts. As mentioned before, the decentralized and public structure of open

[3]Wei is the smallest denomination of Ether.

blockchains necessitates mechanisms such as PoW or PoS that ensure trust between users. These consensus mechanisms are the reason why blockchain technology offers extremely good security (Shetty, Kamhoua, and Njilla, 2019). As such, Ethereum offers an ideal environment to implement methods that ensure the data-sharing challenges established in section 2.1 do not pose an issue.

In chapter 4, we will discuss how our platform ensures compliance with the GDPR in detail.

2.2.2 Smart Contracts

Without a broad range of applications for the blockchain environment, the technology would not be as powerful as it is. Ethereum is based on a programming language called Solidity (section 2.2.4) and every transaction is built on so-called smart contracts that allow any person to freely design transaction systems on the Ethereum blockchain. There is a wide variety of different projects currently in development, ranging from voting systems to employment contracts, supply chain management, and the protection of copyrighted content.

All of these are based on transactional logic defined in smart contracts.

A smart contract is a digital protocol that facilitates, verifies, and executes one or multiple transactions (Rosic, 2016). The type of transaction can vary strongly. Since smart contracts are deployed to the blockchain the inherent transactions benefit from its trustless environment because it is impossible to make unauthorized changes to the contract after deployment. This means the actual function code and its internal mechanisms are immutable, while its variables are not. The only way to change the values of variables is through the specific pre-defined functions inside the contract, which are accessible only with the correct credentials associated with the individual's respective public key (here synonymous with their obscured identity).

Since the only way to make changes to the smart contract is through predefined functions, a poorly defined function in a smart contract could lead to serious security risks, which makes in-depth testing a necessity (Luu et al., 2016). A correctly defined function in a smart contract will disallow unauthorized users from making the respective transaction, and this is incontrovertible. The security of a smart contract is thus at the discretion of its creator.

For this thesis, we developed smart contracts that define the business logic powering the LUCE platform. This results in high security due to inherent blockchain technology (immutable and trustless), high flexibility due to the nature of smart contracts (self-executing, predefined logic), and strong tools for monitoring transactions (public ledger).

2.2.3 Ethereum Tokens

Ethereum tokens are a special sub-type of cryptocurrency, usually defined as fungible, exchangeable assets. They are created from specialized smart contracts and are mostly used to create secondary economies on top of the Ethereum network. A noteworthy example of this is the DAI stablecoin (Tran, 2020), which is based on the ERC-20 token standard (Vogelsteller and Buterin, 2015) and is perfectly robust against the volatility that other cryptocurrencies such as Ether or Bitcoin commonly experience[4] (ERC, Ethereum Request for Comments). As a sub-currency of Ether, it is traded on the Ethereum network.

The Ethereum Improvement Proposals (EIP, Becze and Jameson, 2020) are a collection of standards, new features, or updates for the Ethereum network. The ERC-20 token standard is one of these, and arguably the most famous one due to the success of DAI and others. The main advantage of these standards is a platform-wide standard practice for method definition (which leads to fewer fatally faulty contracts being deployed) and the resulting easy implementation of interoperability of contracts.

Different adaptions of these token standards can represent a wide range of assets. This is why most advanced smart contracts that represent assets are based in some way or form on token standards as defined in the EIPs.

The goal of using tokens in the LUCE platform is to provide a sophisticated system that allows automated access control to datasets and serves as an identifier for how specific data requesters interact with specific datasets, such that the supervisory authority can inspect these interactions in case of a complaint. In particular, this will be useful in solving the data-sharing challenge C2 of access exclusivity, since tokens

[4]Catania, Grassi, and Ravazzolo, 2018 investigate the volatility of several cryptocurrencies and create an econometric prediction model, after finding several similarities between cryptocurrencies and other financial time-series.

allow for intricate control over who can and cannot access a dataset. For this purpose we investigated several different token standards, as shown below:

ERC-20 is a standard API for tokens in smart contracts that provides base functionality to transfer tokens or approval for third parties to transfer tokens. It has the widest base, meaning it can theoretically interface with many existing smart contracts. Unfortunately, there is no mechanism to protect against faulty token transactions, making them irrecoverable in certain cases.

ERC-165 is an interface standardization for implementation and identification of interfaces for smart contracts. It can detect if a contract implements an ERC-165 interface or any other interface. With the implementation of this standard, we can avoid calling unsupported functions or faulty input values, which supports the interoperability of smart contracts. However, it has no standalone use, i.e. must be used in concert with another token standard.

ERC-721 is based on ERC-20 but implements a token standard where each token is unique and can have different values (non-fungible). This makes it useful for representing physical property and other such assets. ERC-721 tracks ownership of each token individually. Additionally, tokens can be deleted and associated methods are robust against faulty inputs. However, it does not provide any type of data structure to associate tokens with individual properties.

ERC-777 is based on ERC-20 and backward compatible. It implements methods to allow token transfers on behalf of another address, meaning it allows for flexibility in terms of *gas* costs allocation for transactions. Unfortunately, its tokens are not unique.

ERC-1337 introduces a standardized subscription method for periodically occurring transactions. It implements a 'value stream' that periodically transmits the value to a designated address. However, it requires staking the subscription amount from the beginning, as well as additional mini-fees for miners to transact.

In this thesis, we adapt the ERC-721 token standard to represent a unique access key to specific datasets, because it is the closest standard to what we wish to achieve with our token implementation. In section 3.5 we will explain the reasoning behind this in more detail and in section 4.3.1 we will explain our technical implementation, including how we add the data structure that will include requesters' meta-information such as their license.

2.2.4 Solidity

Solidity is an object-oriented programming language designed for creating smart contracts on the Ethereum platform, as well as several others. Solidity was developed specifically to allow programmers to create applications that enforce business transactions based on the logic specified in the underlying smart contract.

A contract written in Solidity consists of two parts; the code that represents its functions, and the data that is stored within its variables, mappings, and structures. This contract is associated with a specific address on the Ethereum blockchain, which is publicly available and allows anyone with the correct credentials to call the functions within the contract.

2.3 Incentive Types

This section describes the incentive types that are relevant to a data-sharing platform. Generally, there are two parties that must be motivated to participate and which we must differentiate: The data provider and the data requester. We must also differentiate the boundary conditions of data-sharing between incentives and requirements for the platform. If we refer back to section 2.1, where we defined the challenges of data-sharing, requirements are to secure and enforce data ownership rights (Challenge C1), as well as providing methods to ensure only authorized individuals can access the data (Challenge C2). These are requirements from the perspective of the data provider. Since LUCE is a blockchain application, generally high security and transparency form a valuable basis in order the tackle both these challenges. Existing platforms create a high level of centralization of data, which is problematic where individuals do not wish to share because it adds multiple layers of complication (due to third-party involvement) to allow these individuals to exercise their rights given by the GDPR. This problem is alleviated by the promise of decentralization and high flexibility offered by public blockchains in combination with smart contract logic. In chapter 3, we will explain the architecture of LUCE, which is designed to provide all necessary tools to enable easy GDPR compliance, while chapter 4 will show the technical realization of the smart contracts that represent these tools. The remaining challenge, compensation, is where incentives can be applied in various forms (Challenge C3).

For data providers, compensations can be **monetary** (economic incentive) or **reputation**-based (social incentive), in addition to the general, intrinsic and intangible benefit that is likely to result from supporting medical **research** through data-sharing. However, there are also aspects of the system that may disincentivize data providers, namely, unavoidable **costs** arising from the usage of our platform.

The requirement from the perspective of data requesters for using the platform is data-availability, which we provide by incentivizing data providers to use the platform. Incentives for the data requester are **inexpensiveness**, **data-quality**, and **data-variety**, all of which support the advantages of data-sharing listed by Tenopir et al., 2011.

Research: Through the intrinsic high value of data, and especially the type of data stored in EHR systems (i.e. medical data that is necessary to perform medical research), the data requesters' requirement for the platform is covered by its purpose. This ties into the general main incentive of the platform, which is the intangible consequence of promoting data-sharing on a large scale. In the field of medical research, this incentive is uniquely powerful due to the aforementioned potential benefit to all humans resulting from research on shared data (though similar effects may be observed from other quality-of-life improving research). By providing a large range of datasets available for specific research purposes (i.e. **data-variety**), a platform such as ours could substantially increase the speed of medical innovation, which will arguably result in a proportional, if more gradual, improvement to global welfare. This positive effect on welfare is very hard to measure and thus rarely represents a compelling incentive for the general population. This is an important factor, since data subjects, i.e. the general population, must first give consent to their data being shared and are unlikely to do so if there is no tangible direct benefit to them, while at the same time, usage of the platform still incurs costs. Thus, neither the data subject nor data provider is unconditionally incentivized to use the platform, which means the platform must employ adequate measures to placate the issue of **costs**.

Monetary: The most obvious incentive for data providers is monetary compensation. If a data provider (and possibly also the data subject) can directly profit from sharing their data, they will usually be more likely to do so than if they cannot. There are also arguments against monetary incentives for medical data-sharing, e.g. if a data subject or data provider can profit from sharing data, they are incentivized to overlook or ignore important consequences resulting from doing so, such as the risk of data being leaked[5] (which our platform protects against through legally binding contracts and precise, immutable records of the data-sharing process), or negative new insights on personal health that lead to significant decisions affecting the person's life (e.g. learning of a predisposition to a disease that might affect them[6]). Since the demand for medical data is very high, a self-regulating market would quickly reveal what types of data are profitable and what types are not, especially if data requesters must pay the providers' compensation. This could theoretically lead to a lack of supply of extremely specialized data for which the demand is not high, thus sharing it may not be profitable, and could effectually result in common research areas being prioritized over others. However, in chapter 6 we can observe in our model and a 'profitable' scenario that the cost of publishing a dataset is covered after the fifteenth data requester pays their respective fee. Thus, the sixteenth data request already generates profit. This puts the data provider in a unique position because the entry cost into the market is quite low (as we show in section 6.2), and even niche biomedical research can be easily supported, which once more supports the data requesters' incentive of **data-variety**.

Costs: Both data providers and data requesters are disincentivized to use the system if it is expensive to do so. If we compare the current high cost of collecting data (Getz et al., 2015) with our results in chapter 6, then the cost of using the platform is negligible for data requesters. This is not the case for data providers. Both to share data and to update it later (in compliance with GDPR requirements),

[5]Such risks mainly stem from re-identification of anonymized data (especially a risk where genetic information is concerned (Erlich, Pe'er, and Carmi, 2018)) unintentionally falling into the hands of third parties such as insurances or employers.

[6]The American College of Medical Genetics regularly releases recommendations regarding if, and to what extent incidental findings resulting from genetic sequencing, that go beyond what the patient specifically asked for, should be shared with the patient (Green et al., 2013). This is problematic because patients might misinterpret the lack of specific findings as evidence that they were 'tested' and found 'healthy' in those aspects. Conversely, if all incidental findings are shared, patients might misinterpret these as representing a certainty that they will be affected by a certain health condition.

providers must make initial investments, which is due to the underlying technology of the platform. Any transaction made on the blockchain (e.g. an update to the data) incurs a cost, which the transactor must pay[7]. This necessary initial investment serves as an entry barrier for poor-quality data because providers will be less willing to take the risks associated with publishing such data, which ties into the data requesters' **data-quality** incentive. However, it is still possible to counteract the provider's cost disincentive while still maintaining the barrier against poor data quality by implementing a weaker form of the monetary compensation described above, meaning each data requester is required to partially cover the costs accumulated by data providers. As such, data providers must make relatively small upfront payments, but would effectively have no costs (see section 6.2). Consequently, if data providers have no costs, then data subjects will not incur any costs from giving consent either, meaning they are not inherently disincentivized to share their data due to financial limitations, while they potentially stand to reap great benefits from **research** performed on that data. Additionally, because data requesters will only partially cover the costs accrued by the respective data provider, their incentive for **inexpensiveness** remains relatively intact, as section 6.2 will show.

Reputation: An incentive that does not directly involve monetary compensation could be reputation, exemplified by initiatives such as the Fair Trade movement, which experience commercial success that stands in contrast to its relative lack of resources (Davis and Ryals, 2010). If a data provider (e.g. a hospital) shares data on the platform proposed in this thesis, they would most likely get mention and recognition through the researcher who uses the data, and consequently in the academic community at large. This, in turn, could theoretically, indirectly lead to additional revenue by attracting new customers (i.e. patients) to their institution. This could affect patients' welfare in various ways, one of which could be the improvement (in terms of density/representation) of the data itself due to the inclusion of new patients. Medical institutions that support data-sharing in such a way could also attract other interested parties, such as the state, pharmaceutical corporations, or insurance agencies. The latter business in particular has a vested interest in the well-being of

[7]There exists a workaround for this as explained by OpenZeppelin, 2020, but it slows down the system and is not beneficial due to our smart contract implementation shown in section 4.2.5, since it disables the notion of requesters partially covering the costs of providers.

its customers. Thus, some form of cooperation of insurance agencies with participating data providers is not an unlikely scenario, even if the data is never shared with the insurances directly. Health insurances benefit indirectly through the potential acceleration of medical **research**, which may result in new medical innovations and, in the long term, generally improved health of the population. On the other hand, pharmaceutical corporations would benefit directly if medical data were ever shared with them (though we currently do not envision the support of non-academic research through our platform). However, they could theoretically also benefit from public research that advances the medical field in general by the invention of new methods. Of course, profit margins would be limited because the respective patents would not be associated with the pharmaceutical corporation in question, but the production of new medicine could be profitable anyway. Additionally, all parties involved might be able to advertise their partial involvement in results stemming from analyses performed on shared data, and data providers could gain an even better reputation in specific fields and research areas as a good data source for future researchers. Furthermore, the possession of specific licenses required to access certain datasets might lead to researchers gaining additional recognition, as these licenses could in the future become an identifying marker representing participation in and support of the data-sharing system, and consequently a positive influence on the respective researcher's reputation.

The only incentives we can realistically simulate, without extensive surveys and practical experimentation in a real-world test environment, are **monetary** compensation and **cost** allocation (see section 5). However, we will still theorize how other incentives could logically affect different scenarios (see section 6.1).

2.4 Related work

To provide an understanding of the data-sharing sector and how LUCE classifies in it, we will first introduce two similar platforms, how they operate and point out their strengths and weaknesses.

2.4.1 MedRec

MedRec was introduced by Azaria et al., 2016. At its core, MedRec is a decentralized version of EHR that uses blockchain technology to facilitate easy control for patients over their medical data. Similar to our motivation with LUCE, Azaria et al., 2016 base their argument for the platform on the fact that the current medical records systems in the United States are outdated and no longer provide adequate support for patient inquiries. Additionally, they also point out the lack of medical research support and the potential worldwide benefit from a safe data-sharing system that allows researchers to access data more easily.

Three actors interact with MedRec. The patient is registered with the system and automatically associated with any transaction they have a stake in, i.e. an update to their data. Thus, patients can easily track how their data is being handled.

The second actor is the medical institution that records the patients' data. They hold a list of access permissions that automatically regulate how any person interacts with the data. These interactions are saved on the blockchain and always include pointers that clearly show the path the data takes, i.e. who it is shared with and to whom it belongs.

The third actor sustains the system through their computational power, which is used to perform the necessary mining tasks resulting from transactions in the system (Proof of Work). In MedRec there is only one type of transaction: An update of some form to the patient data. This may be an addition of new data, the rectification of old data, a change in access permissions to data, or the deletion of data.

Each transaction is made with a bounty attachment, which consists of aggregate, anonymized data concerning some part of patient records, such as specific blood levels of all patients examined in a certain timeframe. When the respective block is mined, the miner's identity is automatically connected to the bounty. As such, sharing data happens as a side-product of transactions on MedRec, which is the inherent incentive for miners to participate in the system.

MedRec has several shortcomings. First and foremost, there is no mechanism for propagating updates to those who have accessed the data. Second, there is no integrated tool that allows for revocation of access to the data. Lastly, the precision of control over who can access specific data is limited at best, since neither patients nor the medical institutions sharing the data can control who mines a specific block and receives the attached bounty. This results in a rather loose definition of patient data ownership compared to that in the GDPR, though to be fair, the GDPR does not apply in the United States (in this case the Health Insurance Portability and Accountability Act is more relevant). As such, MedRec is likely not suitable for the EU area.

2.4.2 Ocean Protocol

Similar to LUCE and MedRec, the creators of the Ocean Protocol recognize the high value of data and seek to establish an ecosystem for data-sharing across all industries, meaning the main target of their service offering is data-exchange between companies. The Ocean Protocol Foundation, BigchainDB GmbH, and Newton Circus, 2019 introduce their platform as a decentralized data exchange blockchain application. They establish a proprietary cryptographic token that serves as currency on the platform and represents the revenue data providers may generate through sharing their data. Apart from the inherent value of data-exchange, this revenue generation is the main incentive for using the Ocean protocol.

On the surface, Ocean functions as a Marketplace that lists all available datasets. However, Ocean does not implement an internal data storage system. Instead, data providers hold the data themselves and only release it when there is a legitimate request, verifiable through a respective entry in the underlying blockchain smart contract.

Their blockchain is permissioned (similar to Hyperledger), meaning only authorized users can interact with it in any way. Its consent mechanism is called Proof of Authority (PoA). In the environment of permissioned blockchains, De Angelis et al., 2017 discuss this mechanism as a performance-based consensus algorithm that relies on a network of confirmed nodes called authorities. The consensus is based on a rotation schema between these nodes that controls the distribution of block creation to authorities. In their paper, De Angelis et al., 2017 conclude that in certain settings other consensus mechanisms outperform PoA, such as the practical Byzantine fault tolerant often used in PoS.

The economy of Ocean is based on their in-house crypto-token called OCN. This token is used as an incentive for miners (block reward) in the network, who can use it as currency on the Marketplace. Additionally, it is used as a tool to ensure the long-term sustainability of the platform by limiting the total supply and allocating a large part to the main stakeholders of the system. Lastly, the OCN token discourages sharing poor quality data by implementing a staking mechanism that ties the provided data to personal assets - high-quality data would then result in reaching the break-even-point quickly (Ocean Protocol Foundation, 2019).

The system design of the Ocean platform comes with several drawbacks. The way the Ocean Protocol Foundation, BigchainDB GmbH, and Newton Circus, 2019 use their in-house token, adds a layer of complication to the system that does not necessarily ensure asset value-retention, since Ocean actively avoided implementation of price stability due to performance concerns. Another drawback is the lack of autonomous tools for the data provider and data subject to directly, effectively facilitate GDPR compliance.

2.5 LUCE

The License accoUntability and CompliancE framework, LUCE, initially proposed by Havelange et al., 2019, is a system that allows users to upload their data while retaining full control over it. Contrary to the Ocean platform, LUCE ensures compliance with the GDPR by giving the data provider personalized methods to control their data. These methods allow data providers to easily change settings that control who has access rights to their dataset, why, and for how long. Additionally, the data provider can issue updates, change the required license, or completely delete the dataset. All of these changes perpetuate through the system.

The LUCE platform consists of two major components. The first component is a web-server bundled with its databases. These incorporate a website as a user interface, which users can use to explore and request available datasets. These datasets are stored on an encrypted cloud platform. The second component is the blockchain on which Ethereum smart contracts are deployed, and the focus of this thesis. These

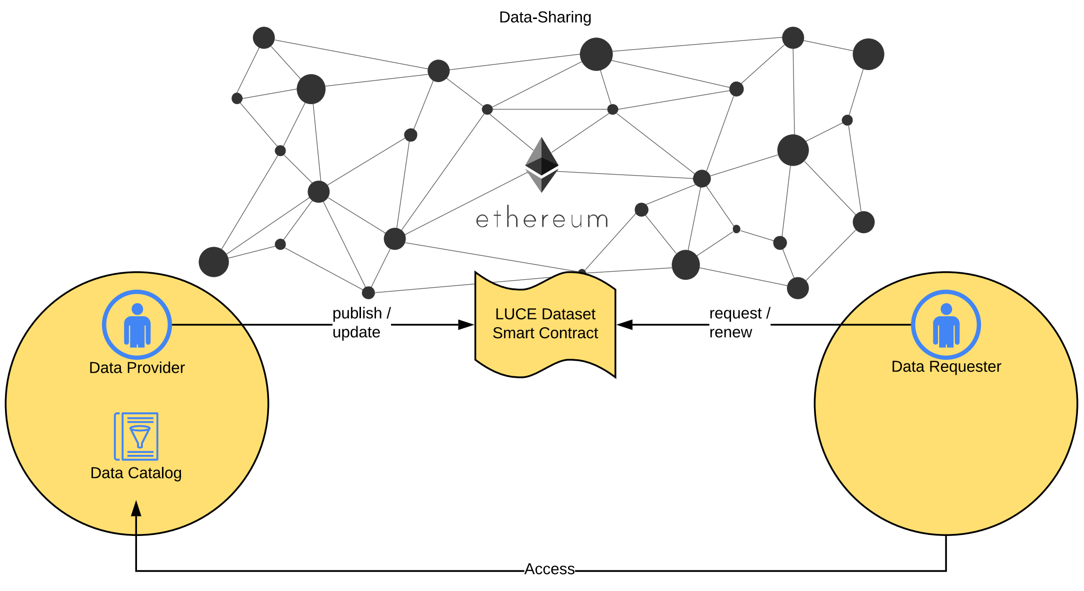


FIGURE 2.1: Original LUCE smart contract architecture. Figure 3.1 shows how we extend the smart contract modules.

smart contracts handle all interactions between users and the data, meaning it is the best interface for potential incentive implementations.

Four parties interact with the platform in various ways - the data subject, data provider, data requester, and supervisory authority. We will give a brief overview of how each party can interact with the system, however, the thesis as a whole will concentrate on the main actors in the system, data provider, and data requester.

Figure 2.1 shows an overview of how these main actors of the system interact with the smart contract logic. A data provider (e.g. hospital) can generally publish and update their datasets. A published dataset appears in the data catalog. When a dataset is published, the data provider must supply information on it according to the Automatable Discovery and Access Matrix (ADA-M). This includes meta-information on the dataset, access requirements, and an access link, all of which are saved to the respective smart contract. Thus, each dataset must be connected to a separate smart contract. This allows the provider intricate control over how each dataset should be accessed by requesters. If a data requester (academic researcher) fulfills the requirements set by the data provider, they can make access requests, which are always time-limited. However, requesters can also renew their access time. GDPR compliance is given through the smart contract logic, which legally binds all requesters to the access conditions of each respective dataset. Jaiman and

Urovi, 2020 developed a smart contract module that controls these access conditions for datasets, applied specifically in the sector of EHR systems. Their approach combines the ADA-M meta-information with the Data Use Ontology (DUO), proposed by Dyke et al., 2016. Their smart contract facilitates precise declarations concerning sharing permissions through the purpose codes introduced in DUO. In simple terms, this translates to a series of requirements concerning the proposed use for the data. As an example, this could mean, that the data provider makes their data available only for disease-specific research concerning exclusively drug development for a certain age group. The smart contract would then enforce these access conditions.

Besides the data provider and data requester, we briefly mentioned two other actors in the system. The data subject (e.g. patient) fundamentally enables the system, even though their interactions with it are very limited, albeit powerful since they are the source of the data and thus hold the ultimate authority over what happens with it. The most important aspect of the data subjects' purview is their right for rectification or erasure of data connected to their person. However, the data subject is not directly involved with any of the actual transactions within the system. Their stake in the system is the data they consent to share via the LUCE platform.

Lastly, the supervisory authority (e.g. governmental institution) is responsible for enforcing the rights of the data subjects and general prevention of abuse of the platform. Should there arise a legal issue, i.e. a data requester's non-compliance with the license agreement of a specific dataset, the supervisory authority is responsible for auditing the related metadata and various system interactions of the parties involved.

Almost all user interactions with the system will take place on the LUCE website, meaning users will rarely or possibly never directly interact with the blockchain and smart contracts. Still, it is technically possible to interact with the smart contracts directly since the Ethereum blockchain is public and transactions only require knowledge of the address of the desired party. This is why we implement access conditions that block unauthorized access attempts. As such, even if an unauthorized individual knew the intricate functionalities of the inherent smart contracts, they would not be able to access any data.

Angerer, 2019a developed a working, reproducible prototype of the LUCE framework proposed by Havelange et al., 2019. However, the smart contract logic of their prototype does not fulfill the requirements for data-sharing. In particular, it fails to implement any methods to ensure access exclusivity (Challenge C2), and oversimplifies most aspects of data ownership (Challenge C1). This is why, we extend the prototype built by Angerer, 2019a to include mechanisms to tackle both of these challenges and implement methods for monetary compensation as well. Chapter 3 will show what new modules we add to the smart contract logic, for what purpose, and what they contribute to the system.

Chapter 3

Architecture

This chapter will explain what extensions we make to the LUCE platform to implement incentives. The only incentives or disincentives we can reasonably, realistically model in without extensive surveying and real-world testing are the **costs** resulting from using the system, and how **monetary** incentives may be applied. Before we can implement mechanisms to control these incentives/disincentives, we must first improve upon the underlying system in terms of data control, privacy, internal logic, and access exclusivity. Without providing these prerequisites we cannot accurately model interactions between the actors of the platform since costs are directly linked to the sophistication of the platform. This means, if the underlying smart contracts are oversimplified, the resulting costs will not represent a realistic scenario. This is why we introduce several new modules that elevate the level of sophistication of the smart contract to bring the LUCE platform closer to real-world applicability.

Figure 3.1 shows which system modules we add, compared to the original LUCE smart contract system shown in figure 2.1:

- *Registry Smart Contract* - provides authorization for data publishing and access requests.
- *Dataset Smart Contract* - handles data publishing, updates, cost control, and meta-information (ADA-M).
  - *Smart Contract Ownership* - defines the connected contracts as owned by the data provider that deploys the main contract and is connected to an additional module that allows the owner to delete their smart contract.

- *Access Smart Contract* - handles access and access renewal requests by data requesters and is connected to the ERC-721 token generation contract.
  - *ERC-721 Smart Contract* - adapted token standard that handles the token logic that is key to accessing the data.

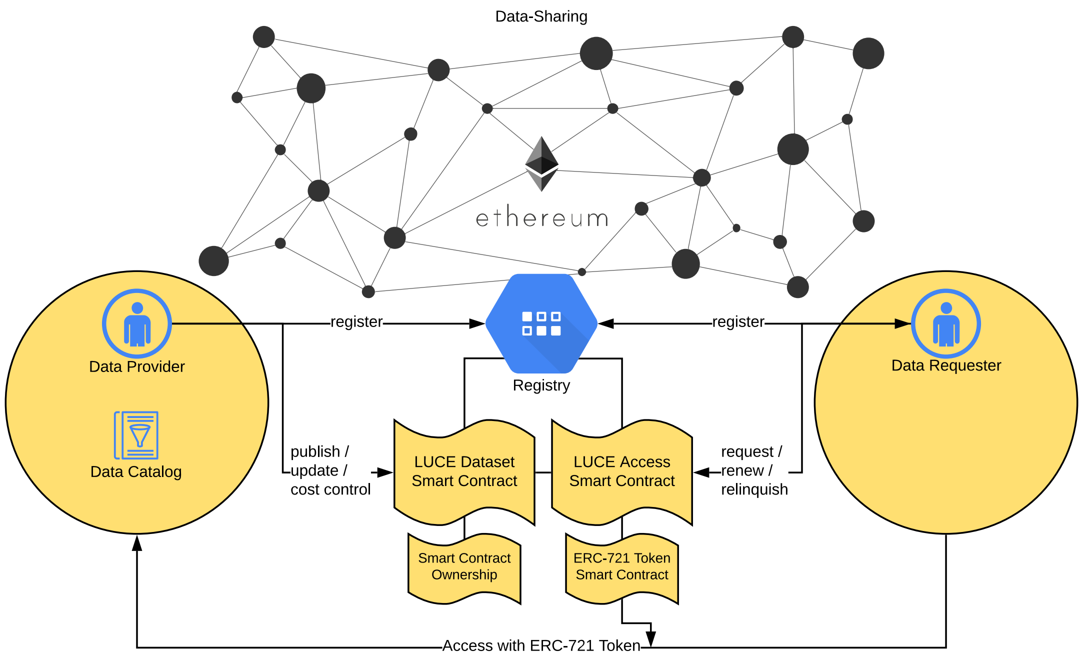

Data-Sharing
ethereum
Data Provider
Data Catalog
register
Registry
register
Data Requester
publish / update / cost control
LUCE Dataset Smart Contract
LUCE Access Smart Contract
request / renew / relinquish
Smart Contract Ownership
ERC-721 Token Smart Contract
Access with ERC-721 Token

FIGURE 3.1: Extension on the framework shown in figure 2.1, detailing what modules we add to the system to improve sophistication.

In the following sections we will explain the reasoning behind, and architecture of the new system modules we implement to improve the LUCE platform.

3.1 Registry Smart Contract

The Ethereum blockchain is public. This means the only conditions for accessing the network and making transactions are an internet connection and knowledge of how to interact with the network. Since data is extremely valuable, and the only barrier to accessing the data shared on LUCE is the underlying smart contract logic, the public nature and transparency of Ethereum make it difficult to limit access to only individuals who should have access. This is why we propose a global registry smart contract that can interface with the LUCE smart contracts to provide access exclusivity (Challenge C2).

This registry is deployed and controlled by the institution responsible for verifying a registrant's information. When a user registers, their information is connected to a wallet in the blockchain, meaning they are effectively anonymous, yet unambiguously associated with their valid license information. As explained in our section on blockchains (2.2), a wallet is essentially a pair of public and private keys[1]. Thus, a user's public key is synonymous with their identity, and, since it is impossible to deduce the identity of the owner from a public key, they can act anonymously. The only information associated with these public keys is the requester's license or the provider's publishing permission, and the only parties privy to identifying information are the owner of the key and the authority that verified the owner's identity.

When an individual makes their first transaction on the blockchain, e.g. publishing a dataset or requesting access to a dataset, their registration information is verified internally. This ensures that no unauthorized individual can interact with the relevant smart contracts, even if they possess the knowledge to circumvent the LUCE website.

However, there are also negative aspects associated with this solution. Since this registry module will be controlled by a centralized institution, we are inadvertently introducing a certain measure of centralization to the LUCE platform. However, we take measures to strongly limit the power of this controlling institution. It has only one purpose: The authentication of users on the LUCE platform. As such, it has only one interface with the smart contract logic, which is to confirm or negate authorization for publishing data or requesting access to data. As such, this centralized control structure functions only as a gateway to the platform but will have no influence on the actual data-sharing process, any possible monetary transactions, or even any purview of how the platform is used. As such, we believe the decentralized nature of our platform remains intact.

As of this thesis, the registry smart contract is incomplete because we have not yet integrated it with an institution that will have sovereignty over it. As such,

[1]Control of the public key and thus all transactions made in their name is completely reliant on their possession of the associated private key. Thus, the entire system is completely dependent on private keys remaining private. Otherwise, the system could quickly fail. There are possibilities to avoid such a scenario, and we implement one of them in our proposed improvement to the LUCE registry, explained in section 4.1.

currently, anyone can register. Refer to section on future work (8.1) for more on this.

3.2 Dataset Smart Contract

The dataset smart contract establishes control for the data provider over their dataset. Each dataset must be published on a separate dataset smart contract. This allows both intricate individual control and automated large-scale control via an external application such as a Python script. This also provides the supervisory authority with a comprehensive records structure, which should make investigations following potential complaints quite simple.

When a data provider publishes their data, they effectively write its meta-information according to ADA-M to the blockchain. This means contextual information on the data, license and purpose requirement information, and a link to the actual data. The data provider can update this meta-information whenever the need arises. The purpose requirement relies on a different smart contract module that was developed by Jaiman and Urovi, 2020 and is specified according to DUO. They introduce a dynamic consent control system to ensure data providers have intricate control over what their data is used for. However, as of the writing of this thesis, their dynamic control smart contract is not yet integrated, which is why we use abstraction to represent purpose codes in our current smart contracts. Figure 3.2 shows an overview of the methods data providers have at their disposal, as they are described in this section.

Still, the meta-information can be changed via an update. Due to GDPR requirements, each update that results in a change in the meta-information of the respective dataset requires all active data requesters to confirm their compliance. As such, they will be notified of the update, and until they have updated their own copy of the data and confirmed this via a special compliance function, the respective requester cannot make access requests to the data.

A different type of update is if the data provider changes the required license to access the data. In this case, all tokens with the wrong license type will be deleted

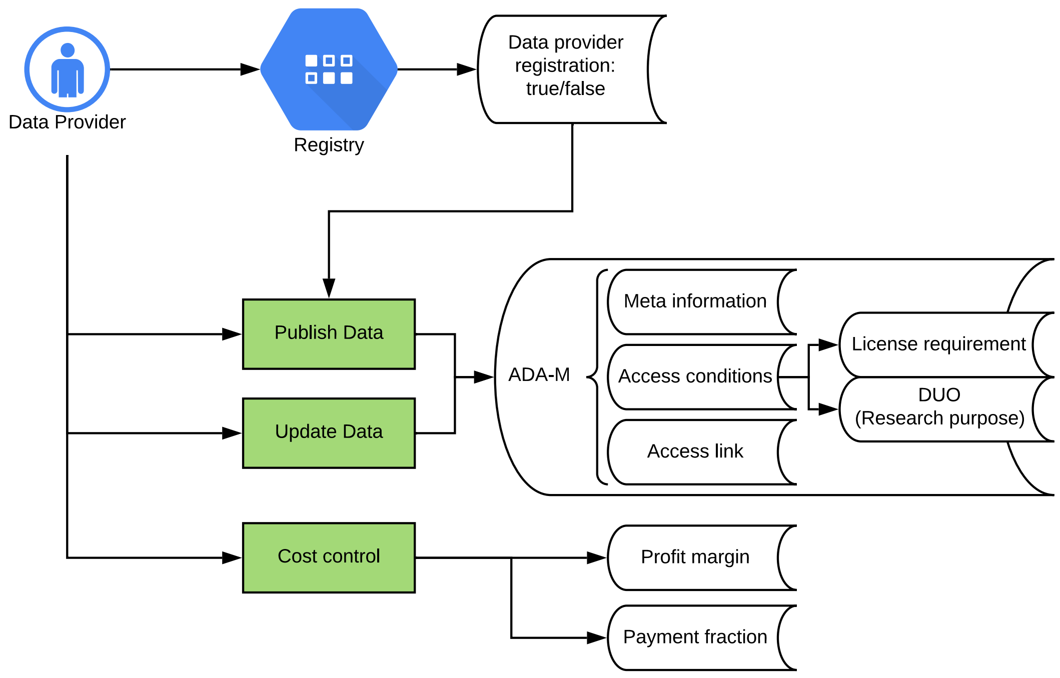


FIGURE 3.2: This is a simplified representation of the control methods a data provider has in the dataset smart contract module.

by the system, and data requesters notified of this. All affected data requesters must then confirm their compliance with this change, and delete their copy of the dataset.

Finally, the data provider is able to establish how the contract handles arising costs. In line with the **monetary** and **cost** incentives as established in our section on incentive types (2.3), there are three possibilities. These will henceforth be known as scenarios:

Scenario 1. No compensation - each party pays only their own arising costs.

Scenario 2. Cost compensation - the data provider's costs are covered by the data requesters.

Scenario 3. Profit - the data provider seeks to profit from sharing their data.

A more detailed explanation of each scenario and their inherent implications will be presented in chapter 6. Generally, the scenarios are meant to showcase how the system reacts to different incentives being implemented. As such, scenario 1 represents no incentives apart from those naturally arising from using the system, meaning data providers are most likely disincentivized from using the system since they incur costs by using it. Scenario 2 seeks to remedy this by implementing a structure that asks data requesters to pay a fraction of the provider's total running costs at the time of their request. Finally, scenario 3 shows how profits may be generated, and how soon the break-even point is reached.

For the purpose of testing these three scenarios, the dataset smart contract allows data providers to manipulate settings regarding cost allocation. They are able to set three variables to control the price data requesters must pay in return for access. First, they can set a percentage profit margin that describes the total earnings aimed for. As such, if the data provider wishes to pay their own costs, it is equal to 0%. For simple cost coverage, it is equal to 100%. If they wish to make a 20% profit, it is equal to 120%. Additionally, the data provider can set the fraction of the total costs/profit they wish each requester to cover. If this fraction evaluates to 0.5, each data requester will pay 50% of the running cost/profit total at that point in time.

3.3 Smart Contract Ownership

This is a simple but powerful smart contract module that establishes a method to control which individuals (i.e. public addresses) can call certain core functions of the underlying contracts, such as issuing an update to the data. When a data provider deploys their copy of the template smart contract to publish a dataset, their address is immediately noted as the owner of that smart contract, and all smart contracts that inherit[2] it.

Arguably the most important function needing authorization of the owner is the destruction of the contract and all super and subordinate contracts. This function is implemented in a smart contract sub-module, which allows the owner to send all funds from the internal balance of the smart contracts to their public address while setting all internal variables to zero. This means any subsequent call to this contract will be voided. With this functionality we can achieve a clean separation between old versions of contracts and newer ones, thus taking steps to prevent accidental misuse of an old contract. With this, we implement the data providers' right to delete their data (GDPR, Article 17). However, it is important to make sure requesters are adequately informed of this change since they could otherwise mistakenly transfer

[2]Inheritance means all non-private variables stored in a particular contract can be called upon by superordinate contracts, but not subordinate contracts.

funds to the destroyed contract, which would result in those funds being lost forever. This is a design choice of the EVM and cannot be programmatically circumvented by any means (chriseth, 2020a). Thus, it is extremely important to ensure the system adapts to this change and disallows any future transactions to this contract. This is why the LUCE website will always automatically delist a deleted dataset's contract address from the *data catalog*.

3.4 Access Smart Contract

This contract handles the main business logic of the LUCE platform. It holds the methods for data access and access renewal requests, implements cost coverage and GDPR compliance systems, and allows data requesters to relinquish their access if it is no longer needed.

Whenever a data requester makes an access request, this contract establishes a connection with the LUCE registry to confirm their license. This is also where we implement the cost coverage system, which applies depending on the settings controlled by the data provider. If all access requirements are met[3], the contract will generate a unique token via the ERC-721 contract, which will be explained in the next section. This unique token serves as an access key for the data requester to the data. Figure 3.3 shows an overview of the methods data requesters have at their disposal, as they are described in this section.

When the data requester successfully gains access to the data, by default they are granted two weeks of access time, after which they must either actively delete their copy of and access to the data, or renew their access time. We implement methods for both options.

Access time renewal necessitates that the data requester has actively confirmed their compliance with GDPR requirements following a potential update by the data provider. The compliance function is quite simple, in that it signifies that the requester that calls it has actively confirmed their compliance with all past updates.

[3]See the technical implementation in section 4.3.1 for all requirements.

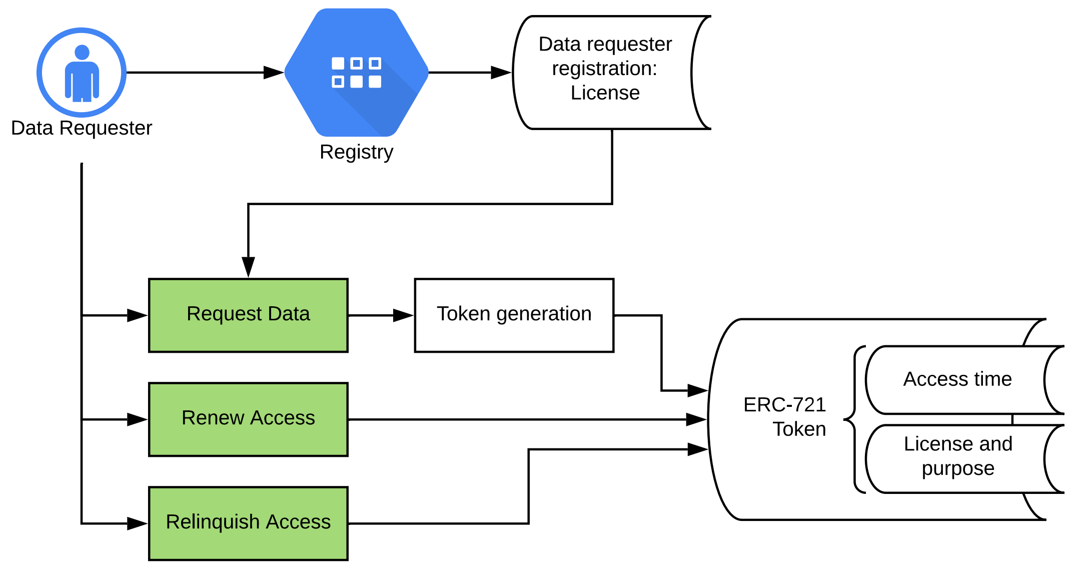


FIGURE 3.3: This is a simplified representation of the access methods a data requester has in the access smart contract module.

This serves as a marker for the supervisory authority should there ever be a complaint against the respective data requester that requires investigation. If this requirement is fulfilled, the data requester is given more access time.

Finally, if the data provider wishes to relinquish their access to the dataset, they can do so by disassociating their public address (i.e. anonymized identity) with the token. This causes the respective data requester to lose access to the data unless they decide to make a new access request.

3.5 ERC-721 Smart Contract

The purpose of giving out tokens as access keys to datasets is that they represent a fixed, standardized data structure that can be easily interfaced with. For this, the token must supply several properties: It must be unique, provide adequate control methods and internal data structures, and be easily traceable. As mentioned in our section on Ethereum tokens, the most fitting candidate is the ERC-721 token standard.

The ERC-721 smart contract module establishes a list of all tokens generated. Factually, a token is simply an entry in this list, represented by a unique ID that unambiguously identifies it. This ID is associated with an *owner*, i.e. the individual (public address) that minted it. Only the owner can transfer the token to another individual. The transfer of a token results in all associated values being accessible and controlled by the new owner. This is not what we wish to achieve with our token implementation, since requesters should not have the ability to transfer their token to other requesters. This is why we created a new structure that associates the token ID with its *user*, i.e. requester. This results in the *user* of a token only having limited control over it, meaning they can use it for three purposes: accessing the data, renewing access time to the data, and deleting their access to the data.

Additionally, we created an internal data storage structure that saves meta-information on the requester and the token (e.g. license, access time, etc.), which only the data provider, respective data requester, and supervisory authority can access. By limiting access to this information we protect the privacy of the data requester.

In the next chapter, we will provide details on the technical implementation of the smart contract modules introduced in this chapter.

Chapter 4

Implementation

In this chapter, we provide details on the technical implementation of the smart contract modules introduced in chapter 3. With this, we take the necessary steps to simulate and measure how different incentives affect the users in the system, according to the three scenarios briefly introduced in our section describing how data providers could control costs arising from using the platform (see section 3.2). In terms of methodology, we used Solidity (v. 0.6.2) to formulate the respective smart contracts, tested them in the Remix IDE (Remix IDE, 2020), and later deployed them to the LUCE platform (Angerer, 2019b), which uses the **web3.py** library to interact with the blockchain test network Ganache, 2020[1]. A link to the GitHub repository with the full function code for this thesis in conjunction with the prototype can be found in Appendix A.

Specifically, this chapter describes the *Registry*, and the core LUCE functions, *Publish Data*, *Update Data*, *Set new License*, *Request Access*, *Renew Access*, and *Relinquish Access* that are implemented in the smart contract modules presented in chapter 3. For all these processes we will show Flow Charts that visualize in detail how the system operates, and pair each operation with the relevant code from the Ethereum smart contracts, to provide details on the technical implementation of the required settings that result from the data-sharing challenges C1 and C2, as well as those that facilitate cost control to additionally incentivize usage of the system (challenge C3).

[1]Ganache is a personal blockchain test software designed for testing and development of Ethereum smart contracts and other applications.

4.1 LUCE Registry

The LUCE Registry is powered by a smart contract that will be accessed by all dataset-contracts to ensure both providers and requesters are registered in the system, and also verify if requesters possess the license required to access a respective dataset. The registration process is shown in figure 4.1.

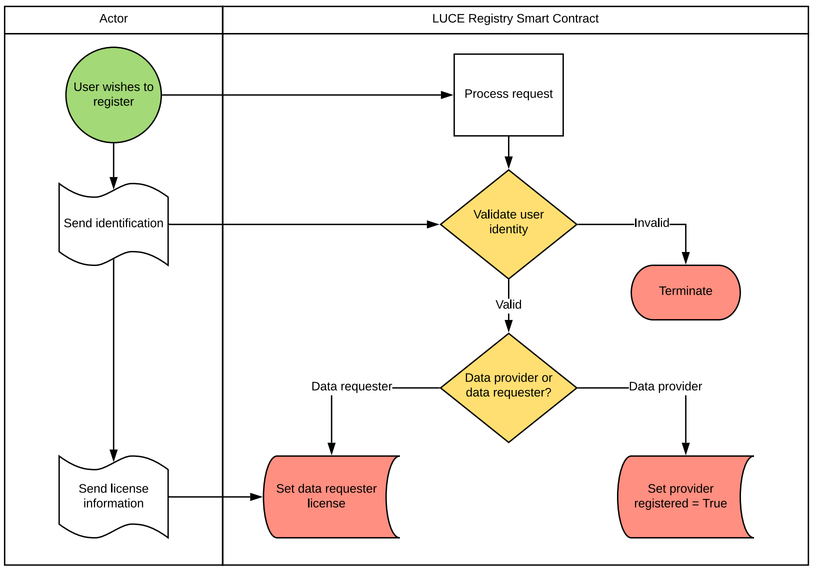


FIGURE 4.1: Logic Flow of a user registering with the system. [Green: starting event, Yellow: requirement statement, Red: end of function]

Corresponding to what we envisioned in section 3.1, detailing the purpose and usage of the registry, data providers are registered with a simple true/false (`bool`) value that is associated with their public key.

```
mapping (address => bool) public providerRegistry;

function newDataProvider(address _provider) external {
    require(providerRegistry[_provider]==false, "This address is
       already registered as Provider.");
    providerRegistry[_provider] = true;
}

function checkProvider(address _provider) external view returns(bool) {
    return (providerRegistry[_provider]);
}
```

When it is necessary to verify the registration of a data provider, the associated contract can simply call up the true/false value associated with their public key (false is standard for all addresses). However, the actual validation process that should theoretically enable registration in the first place is not implemented in the registry smart contract, since it requires interfacing with an existing institution. This will be discussed in chapter 7.

Data requesters will be registered in the same way, the only difference being the value associated with their public key is their license type, here represented by a numerical value. The method and purpose of registration are otherwise the same as for the data provider. The same privacy considerations apply.

```
event newUserRegistered(address indexed user, uint license);
mapping (address => uint) public userRegistry;

function registerNewUser(address newUser, uint license) external {
    require(userRegistry[newUser] == 0, "User is already registered.");
    userRegistry[newUser] = license;
    emit newUserRegistered(newUser, license);
}

function checkUser(address user) external view returns(uint) {
    return (userRegistry[user]);
}
```

The verification mechanism is equivalent to that of the data provider. When it becomes necessary, e.g. upon an access request of a data requester to a dataset, the smart contract calls on the public registry and retrieves the requester's license (standard value for no license is 0).

Should either a data provider's or data requester's private key ever be revealed, the institution responsible for maintaining the registry can deregister the respective user and set the values in the address mappings to false or zero, respectively.

```
function updateUserLicense(uint newLicense) external {
    require(userRegistry[msg.sender] != 0, "User is not yet registered.
        ");
    userRegistry[msg.sender] = newLicense;
}

function deregister() external {
    userRegistry[msg.sender] = 0;
    providerRegistry[msg.sender] = false;
}
```

This would effectively prevent misuse of a revealed private key, and the individual in question could be given a new key pair. In the case of a data provider losing their private key, this should also invoke the deletion mechanism of all associated dataset smart contracts. In the case of a data requester losing their private key, all access tokens associated with their address should be deleted. However, these features are not yet implemented because the registry control authority is similarly not yet installed.

4.2 Data Provider Methods

In the following subsections, we will explain the technical implementation of the core functionalities in the LUCE smart contracts that allow the data provider control over their data. These correspond to the dataset and data ownership smart contract modules introduced in sections 3.2 and 3.3:

- Deploy/destroy smart contracts
- Publish data
- Update data
- Set new license
- Cost allocation control

4.2.1 Deploy smart contracts

When a data provider wishes to publish a dataset, they must first deploy a copy of the LUCE smart contract template to the Ethereum blockchain. In the case of our simulations, we deployed all contracts to the Ganache test network. Through the `owned` smart contract, the data provider is immediately associated with the data provider's public key, meaning the data provider is that contract's owner, as shown in figure 4.2.

The function code below identifies the individual who deploys the contract as the owner of the contract. The `constructor` function is called only once at the deployment of the contract.

```
contract owned {
    constructor() public { owner = msg.sender; }
    address payable owner;
    modifier onlyOwner {
        require(msg.sender == owner, "Only owner can call this function
            .");
        _;
    }
}

contract destructible is owned {
    function destroy() public onlyOwner {
        selfdestruct(owner);
    }
}
```

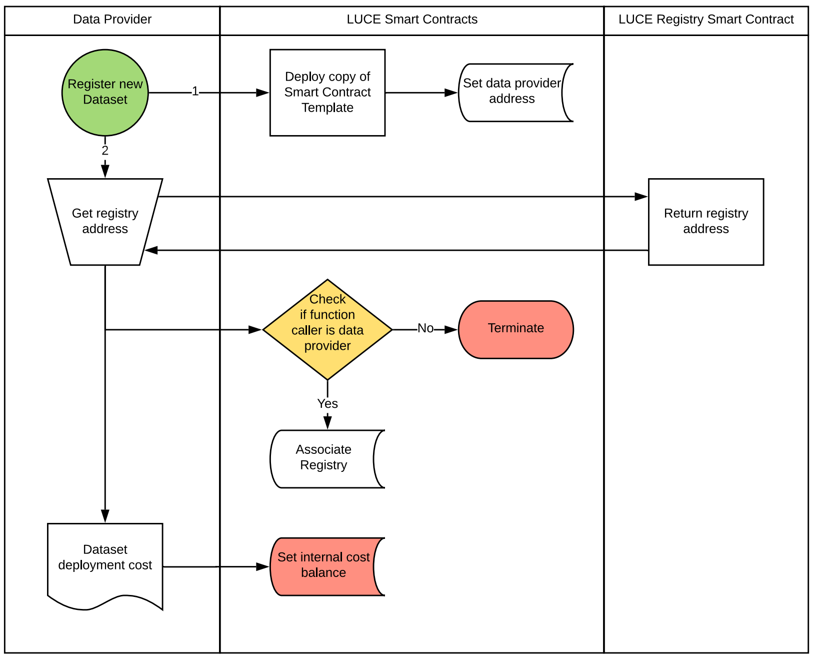

Data Provider
LUCE Smart Contracts
LUCE Registry Smart Contract
Register new Dataset
1
2
Deploy copy of Smart Contract Template
Set data provider address
Get registry address
Return registry address
Check if function caller is data provider
No
Terminate
Yes
Associate Registry
Dataset deployment cost
Set internal cost balance

FIGURE 4.2: Logic Flow of data provider deploying a new contract to enable association with a dataset. [Green: starting event, Yellow: requirement statement, Red: end of function]

The `onlyOwner` modifier[2] can be added to any function definition. This ascertains only the owner (i.e. data provider) can call the respective function. The `destructible` contract inherits the `owned` contract (line 10). This means all non-private functions, modifiers, and variables from the `owned` contract will be accessible. The `destroy` function is modified by the `onlyOwner` modifier, meaning it can only be called by the owner of the contracts. This is the technical implementation of GDPR Article 17, as mentioned in section 3.3 because the main LUCE contract inherits the `destructible` contract.

After deploying the smart contract, the data provider must set the correct registry address so that it knows where to confirm provider and requester registrations (`onlyOwner` ensures that only the data provider can call the respective functions discussed here). In the final version of the contract, this could take place automatically, via a constant that is hard-coded to the contract. For simulation purposes, this was not necessary.

```
address internal registry;

function setRegistryAddress(address userRegistry) public onlyOwner
    providerGasCost {
```

[2]A modifier in Solidity is a method to modify a function to call a specific subcode before and/or after the main function's code execution.

```
    registry = userRegistry;
}
```

In the code above we can see the `setRegistryAddress` function can only be called by the owner of the contract. The `providerGasCost` modifier will be explained in section 4.2.5. Its purpose is to measure the cost resulting from calling the respective function it modifies.

4.2.2 Publish Data

Once the smart contract is deployed, the data provider can publish their dataset. The entire process is shown in figure 4.3.

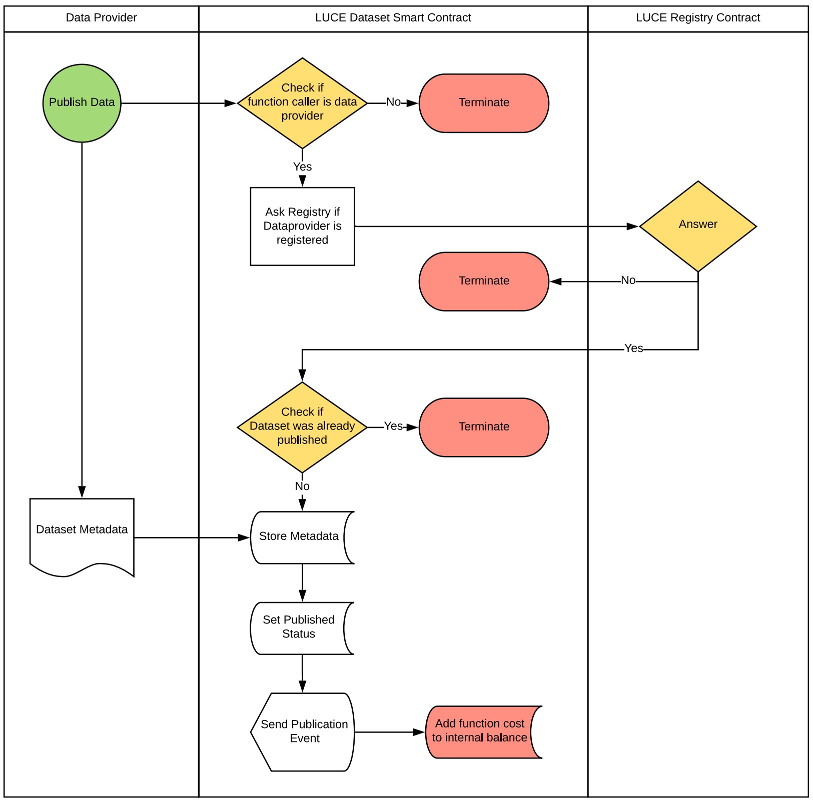

Data Provider
LUCE Dataset Smart Contract
LUCE Registry Contract
Publish Data
Check if function caller is data provider
No
Terminate
Yes
Ask Registry if Dataprovider is registered
Answer
No
Terminate
Yes
Check if Dataset was already published
Yes
Terminate
No
Dataset Metadata
Store Metadata
Set Published Status
Send Publication Event
Add function cost to internal balance

FIGURE 4.3: Logic Flow of data provider publishing a new dataset. [Green: starting event, Yellow: requirement statement, Red: end of function]

Upon calling the `publishData` function, there are two requirements, apart from the `onlyOwner` modifier (line 1). The data provider must be registered (lines 4-6) and the contract must not be associated with a dataset yet (line 2), since a data provider can publish a specific dataset only once. This is to prevent accidentally overwriting an existing dataset with a new one or confusing this functions with the `updateData` function, both of which protect data requesters. The other modifier, `providerGasCost`, measures the cost resulting from calling this function and will be explained in section 4.2.5.

```
1  function publishData(string memory _description, string memory _link,
       uint _license) public onlyOwner providerGasCost {
2      require(unpublished, "Dataset was already published.");
3
4      LUCERegistry c = LUCERegistry(registry);
5      bool registered = c.checkProvider(msg.sender);
6      require(registered, "Potential data provider is not yet registered.
           ");
7
8      dataDescription = _description;
9      license = _license;
10     link = _link;
11
12     emit publishedDataset(msg.sender, _description, license);
13     unpublished = false;
14 }
```

If these requirements evaluate to true, the dataset metadata sent with the function call is stored in the smart contract and it sets its published status to true, followed by a notification to the blockchain that designates the data has been published. The data provider must pass three values into the function (line 1), as defined by ADA-M; a header containing contextual information on the data (here _description), regulatory concepts specifying the required license (here _license), and the link that leads to the data (currently a string, i.e. internet URL[3]). These values are then set to internal variables (lines 8-10).

Finally, the function triggers a notification that logs the publication of the dataset, which can be easily read on the blockchain (line 12) and sets the `unpublished` status to false (line 13).

4.2.3 Update Data

An update to the data can have multiple reasons. Generally, there are two cases. First, a data subject represented in the data requests rectification or deletion of their data. This requires compliance by all data requesters in a reasonable amount of time (e.g. 2 weeks). The second case is an update that adds new observations to an existing dataset. This does not logically require immediate compliance with all data

[3]In later iterations of the LUCE platform, this could be a hash derived from the dataset, which can then be used to find and access said dataset in the secure data storage.

requesters, though new data is an incentive to comply since it could lead to further expedited research.

Figure 4.4 visualizes the process of a data provider making an update to the metadata on the smart contract. Note that the two cases discussed above are indistinguishable from each other regarding the result of the update.

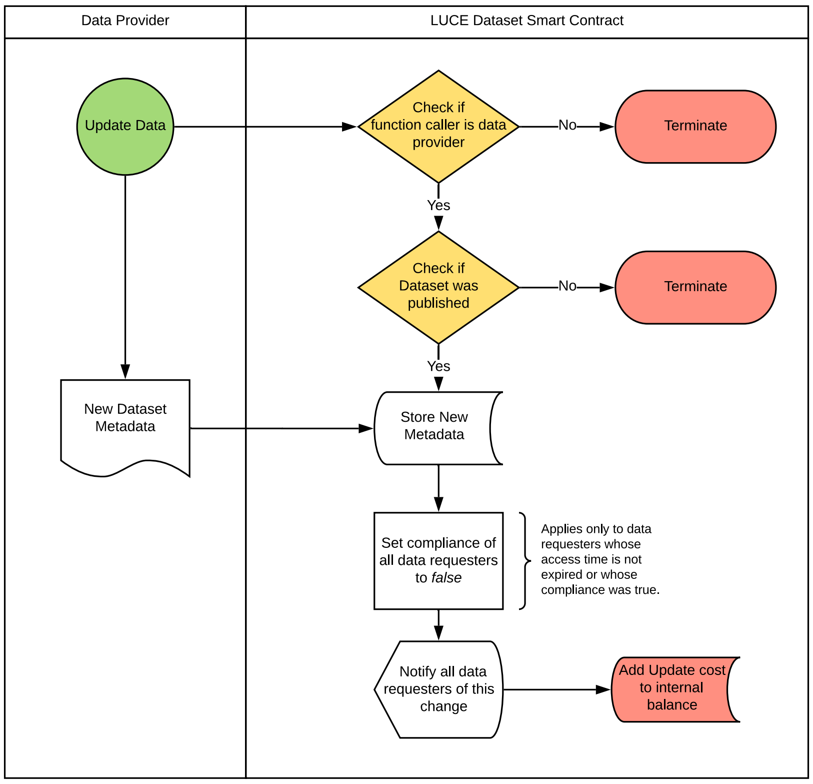

Data Provider
LUCE Dataset Smart Contract
Update Data
Check if function caller is data provider
No
Terminate
Yes
Check if Dataset was published
No
Terminate
Yes
New Dataset Metadata
Store New Metadata
Set compliance of all data requesters to *false*
Applies only to data requesters whose access time is not expired or whose compliance was true.
Notify all data requesters of this change
Add Update cost to internal balance

FIGURE 4.4: Logic Flow of a data provider making an update of their dataset. [Green: starting event, Yellow: requirement statement, Red: end of function]

The two requirements for making an update is that the one issuing the update is the owner of the contract (`onlyOwner` modifier) and that the dataset was published. Once these requirements are fulfilled (lines 1-2), the new metadata fields are set (lines 3-4).

```
1  function updateData(string memory updateDescr, string memory newlink)
       public onlyOwner providerGasCost {
2      require(unpublished == false, "Data was not yet published.");
3      dataDescription = updateDescr;
4      link = newlink;
5      uint arrayLength = tokens.length;
6      if(arrayLength > 0){
7          for (uint i = 0; i<arrayLength; i++) {
8              if (_exists(i.add(1))) {
9                  address to = userOf(i.add(1));
10                 if (tokens[i].accessTime >= now) {
11                     requesterCompliance[to] = false; // This is false
                           until the requester reconfirms their compliance
                           .
12                      emit updateDataset(to, updateDescr);
13                  }
14                  if (requesterCompliance[to] == true) {
15                      requesterCompliance[to] = false; // This is false
                           until the requester reconfirms their compliance
                           .
16                      emit updateDataset(to, updateDescr);
17                  }
18              }
19          }
20      }
21  }
```

Note that an update to the data does not change the required license. A license requirement change invokes different and far more drastic processes that will be explained in section 4.2.4.

Once the new metadata is set, the contract checks all existing tokens (line 8) for the access time associated with them (line 10). If this access time is not expired, the compliance variable of the respective data requesters is set to false and a notification is issued that communicates the update-event across the blockchain (lines 11-12). This event is saved to the logs of the blockchain and can be read via the contract address. The same will happen for those requesters whose compliance variable is still true (lines 14-16), even though their access time expired. In this case, their compliance variable is set to false. This way, we can ignore inactive data requesters in the next update, which potentially saves large amounts of gas if many updates happen in a relatively short period of time.

The update-event notification serves to notify all relevant data requesters that they must update their copy of the data in compliance with the GDPR. Once they do so, they must actively notify the system of their compliance, which is represented by the respective function call to denote their re-compliance. If a data requester no longer needs access to the dataset and deletes their copy of the dataset permanently, they should also delete their access token in the same process, which sets their compliance setting to true one final time. Only then will they be exempt from their legal obligation to follow the data provider's updates, since they no longer possess a copy of or access to the data.

Lastly, similarly to how the `publishData` employs the `providerGasCost` modifier, the `updateData` function will also use it to measure the cost resulting from calling the function.

4.2.4 Set new license

The decision of a data provider to set a new required license for their dataset is drastic because it means all current data requesters will now most likely no longer fulfill the license requirement. Figure 4.5 visualizes the process.

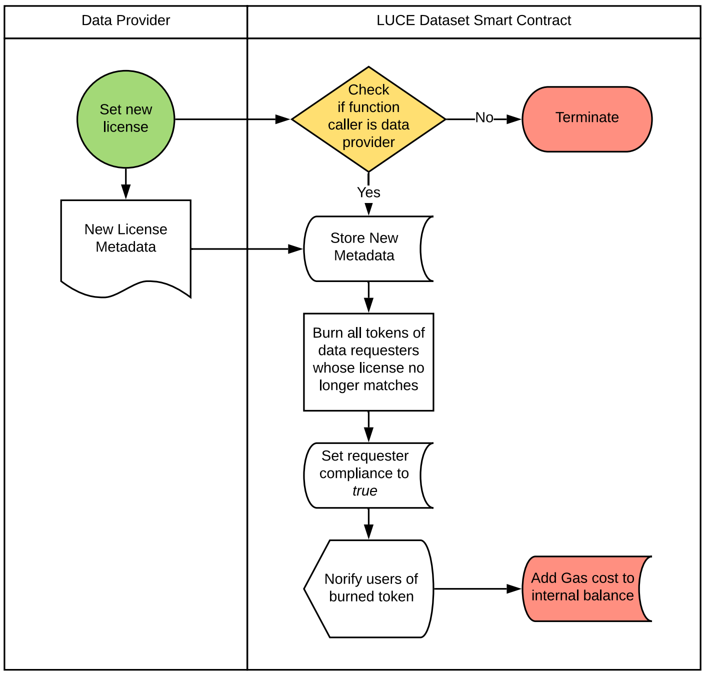

Data Provider
LUCE Dataset Smart Contract
Set new license
Check if function caller is data provider
No
Terminate
Yes
New License Metadata
Store New Metadata
Burn all tokens of data requesters whose license no longer matches
Set requester compliance to *true*
Norify users of burned token
Add Gas cost to internal balance

FIGURE 4.5: Logic Flow of a data provider setting a new license requirement for their dataset. [Green: starting event, Yellow: requirement statement, Red: end of function]

As we can see in figure 4.5, the only requirement for this function is that it is called by the data provider (`onlyOwner` modifier, line 1). If this is the case, the new license value passed by the data provider is stored on the smart contract, replacing the old one (line 2).

```
1   function setlicense(uint newlicense) public onlyOwner providerGasCost {
2       license = newlicense;
3       burnPermission = true;
4       uint arrayLength = tokens.length;
5       if(arrayLength > 0){
6           for (uint i = 0; i<arrayLength; i++) {
7               if(tokens[i].license != newlicense) {
8                   burn(i.add(1)); // Burn all previously added tokens
                        that now have the wrong license.
9               }
10          }
11      }
12      burnPermission = false;
13  }
```

Since this means all current data requesters likely no longer fulfill the license requirement (line 7), all such access tokens are deleted (line 8, refer to section 4.3.3

for the `burn` function code), followed by a notification denoting the public key of the user's access token, the address of the associated smart contract, and their theoretical remaining access time. Additionally, and only in this specific case, the smart contract sets the requester's compliance to false, since the requester's token was deleted by the system and not by the requester, thus compliance must be confirmed separately.

If a data requester somehow changes their license (e.g. they find employment in a different research position which associates them with a different license), they can communicate the corresponding change of their license to the registry contract (section 4.1, `updateUserLicense` function) via the institution responsible for requester verification. If their new license then matches with the new license requirement set by the data provider, they can request access again and receive a new token.

In our current registry smart contract, any data requester can autonomously change their license at any time without consequences. This will only remain the case for the purpose of our simulation since license changes controlled by some separate authority are irrelevant from a cost incentive perspective to both data providers and data requesters, both because these changes are inexpensive and because each individual is responsible for their own registration, as well as changes to their registration. Additionally, the only individual who would ever need to change their license is the data requester, whose costs are not relevant for the data provider.

4.2.5 Cost allocation control

In Ethereum, every transaction is given a gas stipend that is debited to run the calculations resulting from the transaction. The modifier shown below calculates the cost of calling the underlying function and itself, and adds the result to the internal cost balance of the smart contract (line 8). This cost is calculated by saving the total gas stipend before the function call (line 2) and subtracting the remaining gas stipend after the function call (lines 4-5)[4]. The result is added to the constant gas cost of the modifier itself (line 6) and then multiplied by the gas price and the desired *profit margin* (line 7). Since the Solidity language is not built to handle decimals,

[4]Line 3 in this modifier denotes where the main function code should be called, i.e. the function code that is being modified by this modifier.

the *profit margin* must be noted as a fraction. The code below shows the exact steps as described above.

```
1  modifier providerGasCost() {
2      uint remainingGasStart = gasleft();
3      _;
4      uint remainingGasEnd = gasleft();
5      uint usedGas = remainingGasStart.sub(remainingGasEnd);
6      usedGas.add(30700);
7      uint gasCost = usedGas.mul(tx.gasprice).mul(profitMargin).div(100);
8      currentCost = currentCost.add(gasCost);
9  }
10
11 function setProfitMargin(uint _profitMargin) public onlyOwner
       providerGasCost {
12     profitMargin = _profitMargin;
13 }
14
15 function setMultis(uint mult, uint div) public onlyOwner
       providerGasCost {
16     require(mult <= div, "The given fraction is not below 1.");
17     costMult = mult;
18     costDiv = div;
19 }
```

The running cost of the smart contract is thus calculated as follows (lines 7-8):

$$currentCost_t = currentCost_{t-1} + gasUsed * gasPrice * \frac{profitMargin}{100}$$

In simple terms, the running costs in after a transaction are equal to the running costs before a transaction plus the cost of the transaction times the *profit margin*.

As such, the *profit margin* describes the total earnings aimed for, expenses plus returns, and can be set via the `setProfitMargin` function (lines 11-13). If we do not wish to make a profit, it is equal to 100% i.e. 100% of the pure costs of the data provider. This is not the case if we set the *profit margin* higher than 100%, such as 120%, because we effectively 'falsify' the true costs by adding our desired profit to it. This means, if a data provider wishes to generate profits from sharing their data, they must declare their desired earnings as a linear combination of their costs.

In addition to this, the data provider can control what percentage of the running costs each data requester must pay upon access or access time renewal request by calling the `setMultis` function (lines 15-19). This function sets the enumerator and denominator representing that percentage as a fraction. This fraction must be below 1 because the data provider could otherwise abuse the system to generate illegitimate profits. In our section describing requesters' access requests (4.3.1) we explain how this fraction is applied in our smart contracts.

The `providerGasCost` modifier applies regardless of which scenario we are simulating and represents a convenient way for the data provider to keep track of their running costs in all scenarios (see section 6.1 for detailed descriptions of each scenario).

By using this modifier to measure costs arising from publishing data, we essentially ask the data provider to make an initial investment. This is beneficial for several reasons. First, it discourages poor quality data being shared (similar to the Ocean protocol discussed in section 2.4.2). Second, it reduces the complexity of the system by a large margin, since the alternative is employing meta transactions[5], which would allow the data provider to sign a prepared transaction. In the next step, the data requester could then transact the data provider's signed transaction to the blockchain and thus pay the associated gas cost directly. This alternative solution using meta transactions is inferior to our system described in this chapter, since it does not allow for partial coverage of a grand total (e.g. a percentage of the cost of deploying the contract instead of the total).

4.3 Data Requester Methods

In the following subsections, we will explain the technical implementation of the core functionalities in the LUCE smart contracts that allow the data request to access a dataset. These correspond to the access and ERC-721 smart contract modules introduced in sections 3.4 and 3.5.

- Request access
- Renew access time
- Relinquish access

4.3.1 Request Access

In this section, we will explain how a legitimate data requester receives access to a dataset. We distribute access rights via tokens, which are associated with the data

[5]Meta transactions are a special type of transaction that is signed by one individual and then published so that an arbitrary different individual can execute them in the name of the signer (OpenZeppelin, 2020).

requester once their legitimate claim has been verified through the internal logic visualized in figure 4.6.

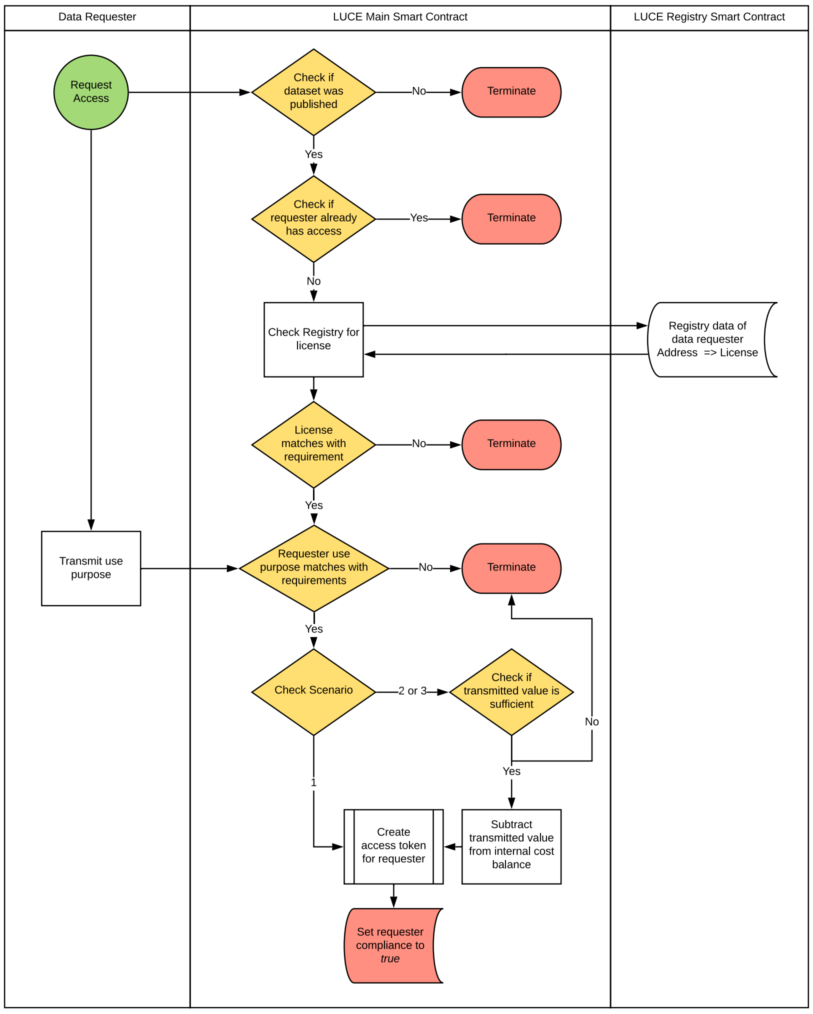

Data Requester
LUCE Main Smart Contract
LUCE Registry Smart Contract
Request Access
Check if dataset was published
No
Terminate
Yes
Check if requester already has access
Yes
Terminate
No
Check Registry for license
Registry data of data requester Address => License
License matches with requirement
No
Terminate
Yes
Transmit use purpose
Requester use purpose matches with requirements
No
Terminate
Yes
Check Scenario
2 or 3
Check if transmitted value is sufficient
No
Yes
1
Create access token for requester
Subtract transmitted value from internal cost balance
Set requester compliance to *true*

FIGURE 4.6: Logic Flow of a data requester making an access request for a dataset. [Green: starting event, Yellow: requirement statement, Red: end of function]

If a data requester wishes to acquire a dataset, a range of requirements must be fulfilled:

- The dataset must be published (line 2). This is to prevent abuse since it would otherwise be theoretically possible to request an access token before a specific license is defined through data publication by the data provider, and could thus possibly circumvent important license terms.
- The requester must not yet own an access token to this dataset (line 11).
- The requester must be registered and possess the same license as is required for accessing the data (line 7).

- The usage purpose confirmed by the requester must match with the requirements the data provider set for this dataset (line 9). For the purpose of this simulation, and until the intricate purpose code formulation by Jaiman and Urovi, 2020 is integrated, the purpose code comparison is represented by a simple arbitrary numerical comparison.
- If all requirements have been fulfilled, the smart contract checks for which scenario simulation we are currently running. If it is scenario 2 or 3 (cost coverage or profit, respectively), an additional requirement must be met: The requester must submit an appropriate amount with their access request (lines 15-17). This amount is calculated by multiplying the current running contract costs with the payment fraction defined by the data provider. This was explained in more detail in our section on cost allocation control (4.2.5).

These requirements correspond to the `require` statements in the function code below.

```
1  function addDataRequester(uint purposeCode, uint accessTime) public
       payable returns(uint){
2      require(unpublished==false, "This contract is not yet associated
           with a published dataset.");
3      LUCERegistry c = LUCERegistry(registry);
4      uint userLicense = c.checkUser(msg.sender);
5
6      // Make sure the requester's license matches with the provider's
           requirements
7      require(license == userLicense, "incorrect license type or user not
            registered");
8      // Make sure the requester's purpose matches the 'requirements' (
           this is where the consent contract will interface)
9      require(purposeCode <= 20, "incorrect purpose Code");
10     // Make sure the requester doesn't have a token yet.
11     require(mappedUsers[msg.sender]==0, "This user already has a token
           and should use renewToken.");
12     addressIndices.push(msg.sender); // Add the data requester to an
           array that lists all requesters.
13
14     // Calculate the amount an individual requester must pay in order
           to receive access and make sure their transferred value matches
           .
15     if(scenario > 1) {
16         uint individualCost = currentCost.mul(costMult).div(costDiv);
17         require(msg.value == individualCost, "Payment does not match
               requirement.");
18         // Adjust the true contract cost by subtracting the value this
               requester transferred.
19         if(currentCost < individualCost) { // Values smaller than 0 are
                not allowed in solidity.
20             currentCost = 0;
21         } else {
22             currentCost = currentCost.sub(individualCost);
23         }
24     }
25
26      // Token generation
27      if(accessTime==0){
28          accessTime = 2 weeks;
29      }
30      _createRequestedToken(license, purposeCode, accessTime); // Creates
            a token.
31      uint tokenId = tokens.length; // ID of the token that was just
           created. Note that solidity is 0-indexed.
32      _safeMint(dataProvider, tokenId); // Mints the token to the
           dataProvider and gives them complete control over it.
33      _safeTransfer(dataProvider, msg.sender, tokenId, ""); // Allows
           access of the created token to the requester.
34      // A requester can look up their token by calling the mappedUsers
           mapping with their own address.
35      mappedUsers[msg.sender] = tokenId; // This proves the requester has
            received a token and cannot receive another one
36      // Compliance initialization for the data requester:
37      requesterCompliance[msg.sender] = true;
38      return tokenId;
39  }
40
41  function expectedCosts() public view returns(uint) {
42      if(scenario == 1) {
43          return 0;
44      }
45      uint individualCost = currentCost.mul(costMult).div(costDiv);
46      return(individualCost);
47  }
```

The amount the requester must pay in scenarios 2 and 3 is a fraction of the running costs of the data provider, as explained in the `providerGasCost` modifier in section 4.2.5. This amount is then subtracted from the total running cost of the data provider (lines 19-22). Note that values smaller than 0 are not allowed in Solidity. Any individual can look up the current expected cost for an access request by calling the `expectedCost` function (lines 41-47).

If this requirement is fulfilled, or in the case of scenario 1 (no payment required), the smart contract then generates an ERC-721 access token minted by the data provider (lines 30 and 32) and transfers usage rights for this token to the data requester (line 33). The access time passed to the function is measured in seconds and its default value is 2 weeks. Solely for the purpose of simulation, we allow requesters to input variable access times (such as 2 seconds), to suitably reduce the time it takes to simulate. In the final version of the smart contract, access time will be a fixed period (such as 2 weeks, to which a requester can add as explained in section 4.3.2). See lines 27 and 28 in the code above for the corresponding smart contract logic.

Once the token is generated and user rights are transferred to the data requester, the last step is to initialize the requester's compliance with the data provider's requirements for using the data (line 39 in the code above). This is mainly important regarding updates to the data since the requester is legally obligated to mirror updates made by the data provider regarding their own copy of the data. This value will be set to false in case of an update, which we explain in the section 4.2.3.

The token generated for the requester is based on the ERC-721 standard of Entriken et al., 2018. All ERC-721 tokens are unique and can have different values. This is why they are called non-fungible. The ERC-721 standard implements an API for non-fungible tokens and thus facilitates easy interfacing with outside contracts. Since it is designed specifically to represent a diverse range of assets such as physical property, virtual collectibles, or even loans or debts, ERC-721 is ideal to represent access keys to datasets.

The token generated for a data requester holds information on that requester, such as their license and purpose code, as well as the time value to denote how long the data requester may access the dataset. Additionally, each token is associated with a unique hash value. The code below shows the data-structure implementation (lines 1-7 and 20) which is associated with each token via its ID and holds information on the specific data requester. We also provide a token hash (lines 12-17) that can be used to validate a token from outside the contract, however, this feature does not yet interface with the LUCE platform in general, which we will discuss in chapter 7.

```
1  struct LUCEToken {
2      uint license;
3      uint purposeCode;
4      uint accessTime;
5      bytes32 tokenHash;
6  }
7  LUCEToken[] internal tokens;
8
9  function _createRequestedToken(uint _license, uint _purposeCode, uint
       _accessTime) internal {
10     uint id;
11     bytes32 lastHash;
12     if(tokens.length < 1){
13         id = 1;
14         lastHash = keccak256(abi.encodePacked(uint(1), msg.sender,
               _license, _purposeCode));
15     } else {
16         lastHash = tokens[tokens.length.sub(1)].tokenHash;
17         id = tokens.length.add(1);
18     }
19     bytes32 tokenHash = keccak256(abi.encodePacked(id.add(uint(lastHash
           )), msg.sender, _license, _purposeCode));
20     tokens.push(LUCEToken(_license, _purposeCode, now.add(_accessTime),
            tokenHash));
21     emit NewToken(id, _license, _purposeCode);
}
```

As mentioned in our introduction of the ERC-721 smart contract module (section 3.5), we make several significant changes to how tokens are handled compared to the standard introduced by Entriken et al., 2018. First and foremost, in our version of the ERC-721 token, ownership can never be transferred to a different address. This means, after its initialization and association with the respective data requester (as its *user*), the token can never be transferred again. The only way data providers can revoke a requester's given access is to change the required license, which causes all access tokens associated with the incorrect (old) license to be deleted (see section 4.2.4). Since data providers can thus not delete specific tokens without affecting others, this measure protects data requesters from targeted discrimination.

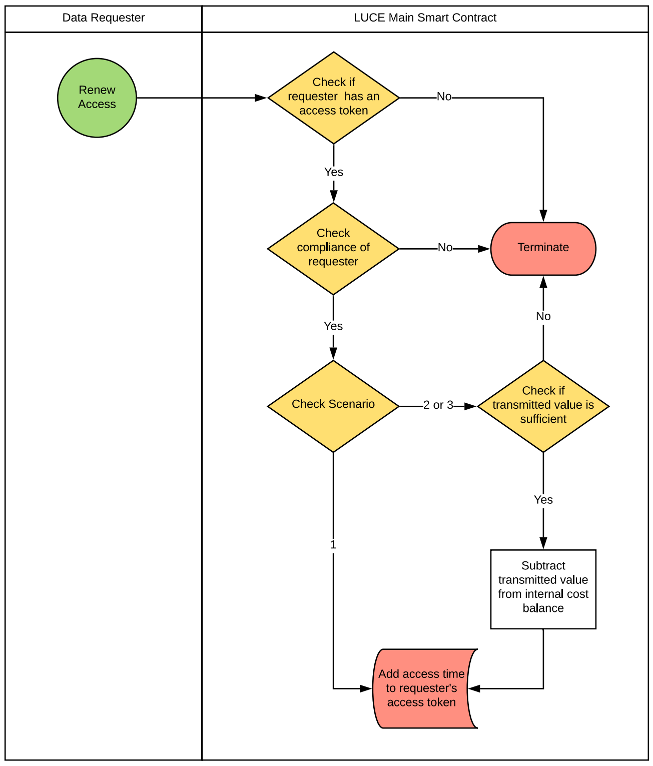
Once the data requester receives an access token, they can call the `getLink` function to download the dataset. Currently, this is a simple, direct link to the dataset stored on the local webserver (line 9). Logically, the data provider can access the link without any additional requirements(lines 2-3). The data requester must have confirmed their compliance with GDPR requirements (line 5, if there was an update to the data), they must possess a token (line 7), and their access time to the data must not be expired (line 8).

```
1  function getLink() public view returns(string memory) {
2      if (msg.sender==dataProvider){
3          return link;
4      }
5      require (requesterCompliance[msg.sender], "Access denied. Reconfirm
           compliance first.");
6      uint tokenId = mappedUsers[msg.sender];
7      require (userOf(tokenId) == msg.sender, "Operation not authorized")
           ;
8      require (tokens[tokenId.sub(1)].accessTime > now, "Access time has
          expired.");
9      return link;
10 }
```

Once the secure data storage is implemented, the internal processes to access data are expected to become slightly more complicated. However, this is extraneous for the data requester, since the process will be automated and the website interface will provide the same functionality as before.

4.3.2 Renew access time

The access time associated with any access token is, outside of the simulation, fixed to a reasonable amount of time (e.g. 2 weeks, line 17). Since access to data may be needed for longer, we implement a function that lets data requesters add to their access time. The process is visualized in figure 4.7.



Data Requester
LUCE Main Smart Contract
Renew Access
Check if requester has an access token
No
Yes
Check compliance of requester
No
Terminate
Yes
No
Check Scenario
2 or 3
Check if transmitted value is sufficient
Yes
1
Subtract transmitted value from internal cost balance
Add access time to requester's access token

FIGURE 4.7: Logic Flow of a data requester renewing their access time for a dataset. [Green: starting event, Yellow: requirement statement, Red: end of function]

There are two general requirements for renewing access to a dataset. First, the requester must have an access token to that specific dataset (line 8). Second, they must have confirmed compliance with any previous updates (line 9). The `confirmCompliance` function (lines 1-4) allows data requesters to notify the system of their GDPR compliance following an update, which allows them to renew their access time to the data.

```
1  function confirmCompliance() public {
2      require (mappedUsers[msg.sender] > 0, "The requester does not own a
           token. Request a token first.");
3      requesterCompliance[msg.sender] = true;
4  }
5
6  function renewToken(uint newAccessTime) public payable {
7      uint tokenId = mappedUsers[msg.sender]; // This defaults to 0 in
          case the requester doesn't own a token. TokenId 0 is invalid.
8      require (userOf(tokenId) == msg.sender, "Operation not authorized");
9      require (requesterCompliance[msg.sender], "Must first comply with new conditions for data access.");
10     // Calculates the value the requester must pay to call this function and checks whether the amount transferred matches.
11     if(scenario > 1) {
12         uint individualCost = currentCost.mul(costMult).div(costDiv);
13         require(msg.value == individualCost, "Payment does not match requirement.");
14         // Adjust the true contract cost by subtracting the value this requester transferred.
15         if(currentCost < individualCost) { // Values smaller than 0 are not allowed in solidity.
16             currentCost = 0;
17         } else {
18             currentCost = currentCost.sub(individualCost);
19         }
20     }
21     if(newAccessTime==0){
22         newAccessTime = 2 weeks;
23     }
24     if(tokens[tokenId.sub(1)].accessTime > now){
25         tokens[tokenId.sub(1)].accessTime = tokens[tokenId.sub(1)].accessTime.add(newAccessTime);
26     } else {
27         tokens[tokenId.sub(1)].accessTime = now.add(newAccessTime);
28     }
29 }
```

If these requirements are fulfilled, the internal logic checks for which scenario we are simulating (line 11). Equivalently to the process when a new data requester makes an access request, there is the same additional requirement in case of scenario 2 or 3. The requester must transmit the appropriate value with their transaction for renewing access time (lines 12-18).

If this requirement is also fulfilled or in case of scenario 1, the standard amount of access time (e.g. 2 weeks) is added to the requester's associated token (lines 24-27).

The code snippet above shows a case differentiation for access time adjustment. Since access time is noted as a precise date measured in seconds from *epoch* (January 1, 1970, 0:00:00 UTC is equivalent to 0 seconds), it is necessary to differentiate between requesters whose access time had already expired (line 27) and those whose access time has not yet expired (lines 24 and 25). Otherwise, requesters might lose access time, which would be unfair.

4.3.3 Relinquish access

The data requester (i.e. *user*) of a token has a limited range of actions they can take, the most relevant of which are accessing the data, renewing their access time to the data, and deleting their token should that ever be required. To delete their token, a data requester must call the `burn` function presented below. There are two ways to call this function. Either the data requester calls it, or the smart contract calls it upon a change in the license requirement (line 2 in the code below). For the latter, see section 4.2.4.

```
1  function burn(uint tokenId) public {
2      require (userOf(tokenId) == msg.sender || burnPermission, "Operation not authorized");
3      address user = userOf(tokenId);
4      uint accessTime = tokens[tokenId.sub(1)].accessTime;
5      uint remainingAccessTime = 0;
6      if (accessTime > now) {
7          remainingAccessTime = remainingAccessTime = accessTime.sub(now);
8      }
9      _burn(tokenId);
10     emit tokenBurned(user, tokenId, address(this), remainingAccessTime);
11     mappedUsers[user] = 0; // indicate the user no longer has a token
12     if (msg.sender == user) {
13         // If the data requester issues deletion of their token, they also intrinsically agree to delete their copy of the dataset.
14         requesterCompliance[user] = true;
15     } else {
16         // Hard compliance must be verified by the supervisory authority, if it is in question.
17         requesterCompliance[user] = false;
18     }
19 }
```

When this requirement is fulfilled, the function first notes the remaining access time (0 if the access time is expired, see lines 4-8 in the code above). Then, the internal `_burn` function of the ERC-721 token standard is called, which associates the token with the null address, meaning it can no longer be used. Regardless of how the function is called, the data requester is notified of the event. If the token deletion was issued by the data requester, their compliance is set to *true* because token deletion should always involve the deletion of the requester's copy of the dataset as well. If the token deletion was issued by a change in the license type, compliance is set to *false*. Refer back to section 4.2.4 for more details on this.

Chapter 5

Simulation

In this chapter, we will provide a detailed description with relevant code snippets that explain how we simulate the different scenarios briefly mentioned in section 3.2[1].

The simulation was developed and run in a Jupyter Notebook in Python. We used the **solcx** package to compile our smart contracts and the **web3.py** package (derived from its JavaScript counterpart) to communicate with the Ganache blockchain test network. Additionally, we used the **pandas** packages to ensure the data generated from our simulation was output in an easily usable format, we used **numpy** for matrix operations and randomization, and **random** for general randomization. For initial visualizations, we used the **altair** package, though we switched to the **Tableau** software to generate the final visualizations shown in this thesis.

The simulation will, in simple terms, allow artificial data providers and requesters to interact with the system over time. As such, they will make transactions on the Ganache blockchain test network, and we will measure the resulting cost.

There are several goals we wish to achieve with this simulation to gain a perspective on how **monetary** and **cost** incentives will affect the platform:

- Measure how costs from using the system arise over time.
- Measure how long it takes for costs to be covered in scenarios 2 and 3.
- Find a balance between cost coverage for the data provider and fair payment amounts for all data requesters.
- Show general applicability of the improved smart contract logic introduced in this thesis.

[1]See section 6.1 for a more detailed explanation of each scenario.

In light of these goals, we will first establish the assumptions we make in our simulations in section 5.1. Next, in section 5.2, we will specify how we initialize each simulation regarding how participants will interact with the system. Section 5.3 defines the input values to our simulations and what they are used for, while section 5.4 defines what output we are generating through our simulation, i.e. what we are measuring. Finally, in section 5.5, we clarify what actions the participants in the system can take, and in section 5.6, we determine the optimal parameters for a realistic simulation.

5.1 Assumptions

We have made the following assumptions about data providers:

- There are far fewer data providers than data requesters.
- The probability of a data provider deciding to publish their dataset is lower than the probability to update it after publishing.
- The probability of both publishing and updating a dataset is constant, independent of consequent potential costs arising, and independent of how many data requesters have access to the dataset.
- The probability of publishing is independent of the type of dataset (e.g. blood levels of patients over the age of 50).
- In scenario 3, data providers set their desired profit margin only once when they publish their dataset, and never change it afterward.
- Data providers do not make use of their right to set a new license requirement for their published dataset.

We have made the following assumptions about data requesters:

- The probability of data requesters taking *action* follows a normal distribution with mean at 50% and standard deviation of roughly 15% (translates to i.i.d. $X \sim \mathcal{N}(\mu, \sigma^2)$ with $\mu = 0$ and $\sigma = 0.1$ before normalization.

- The probability of data requesters taking *action* decreases over time. This means, no data requester will continue to renew access to a single dataset indefinitely.
- The probability of data requesters taking *action* is independent of the number, type, origin, quality, size, and age of available datasets (the probability we associate with data requesters in this simulation is a tool to model the passage of time).
- Data requesters have an unlimited amount of money potentially available to request access or renew access time to datasets.
- The intrinsic value of a dataset never diminishes, no matter how many data requesters have access to it. We will discuss this assumption in chapter 7.
- Data requesters only renew their access time when it is expired, and at the most every 2 periods.

5.2 Initialization

The Ganache software offers the option to generate Ethereum wallets for testing. We chose to generate a total of 1000 accounts to avoid placing technical limitations on ourselves before starting the simulation. This represents a practically endless potential for demand (i.e. potential data requesters) in our simulation. We also supplied each account with 100 Ether, meaning no account would unexpectedly run out of funds.

At the time of this simulation, the value in Dollar of 1 Ether was $316.52 (Chez, 2020). All costs measured in our simulations are derived from this Ether price. Similarly, the *gas* price for an operation on the Ethereum network at the time of this simulation is 72 Gwei (average transaction cost taken from Ethgasstation, 2020)[2].

Our simulation must deliver on two properties:

- A representation of time passing, and

[2]One Gwei (as in Giga-Wei) is equal to 1 billion Wei. Wei denotes the smallest denomination of Ether. One Ether is equivalent to 10^{18} Wei. This means 1 Wei is the smallest possible cost of *gas*.

As an example, a simple transfer of Ether from one wallet to another receives a *gas* stipend of exactly 2,300 *gas*, which is only enough to log the event, and allows no other operation. Thus, the cost of making such a transaction with a *gas* price of 72 Gwei equals $0.052.

- a representation of decision-making (e.g. the decision to request access to a dataset).

Our simulation runs in a loop, repeating itself until a certain requirement is fulfilled. Each iteration of the loop signifies the passing of 1 *period*. In each *period* multiple *actions* can be made. An *action* in this context refers to one of four possible decisions being made: *publishing data*, *updating data*, *requesting access*, or *renewing access time*. We do not simulate data providers changing license requirements because this should be viewed as a last measure, similar to destroying a contract (which is the very last measure). We represent decision-making by randomization. As such, each potential data provider and data requester is associated with a certain probability of taking *action*.

We make the assumption that the chance of data requester taking *action* underlies normal distribution parameters with independent, identically distributed variables, since this is arguably the most commonly occurring distribution in nature:

$$\text{i.i.d. } X \sim \mathcal{N}(\mu, \sigma^2)$$

For simplicity's sake, we center our distribution around 0 ($\mu = 0$) and assume standard deviation is 0.1 ($\sigma = 0.1$).

To associate each account with a normally distributed probability, we first generate 1000 random values of a normal distribution with the aforementioned parameters.

```
x = np.random.normal(mu, sigma, len(accounts))
```

Since the resulting values do not lie between 0 and 1, we normalize them:

```
def normalizeData(data):
    return data - np.min(data)) / (np.max(data) - np.min(data)
```

This results in a vector of random, normally distributed probabilities, which we append to the user accounts list:

```
accounts[['probability']] = normalizeData(x)
```

Thus, a data requester will, on average, have a 50% probability to make an access request in a period. However, since we do no expect data requesters to require access to a specific dataset for an indefinite amount of time, we adjust their probability of taking *action* downwards by a factor of 0.75 each time after they renew their access time to the data.

```
accounts.loc[requester, 'probability'] =
    accounts.loc[requester, 'probability'] * 0.75
```

This results in data requesters renewing their access time only very rarely after the fifteenth time (corresponds to $0.5 * 0.75^{15} = 0.00668$ or 0.668%). Thus we achieve a natural balance of data requesters starting, continuing, and stopping to renew their respective access time and avoid exponential growth of *actions* being taken per *period*, which would be highly unrealistic.

We do not simulate data requesters burning their tokens at that point, since it is irrelevant for the data provider's costs.

For data providers, we assume that the probability of choosing to publish is far lower than for an average data requester making an access request. This is why each data provider is given a uniformly distributed probability to publish that lies between 1% and a maximum probability specified by us (default is 5%). This overwrites the normally distributed probability assigned to the Ganache accounts designated as data providers. This reflects our assumption that data providers are generally less numerous than data requesters and would thus take *action* less often.

Whether or not any user takes an *action* is determined by comparing two numbers: the *action* probability associated with that user and a randomly generated, evenly distributed number between 0 and 1.

5.3 Input values

The simulation offers the following settings. Most of these have default values to make the code easier to read. If we wish to simulate with a different value than the default, we can simply pass a different value to the respective variable in the simulation function.

- `actionTicker`: represents how many *actions* in total we wish to simulate. This controls the length of simulations and represents the exit condition for the loop.

- `costMultiplier`: represents the enumerator of the fraction we use to calculate what percentage of the total each data requester should pay in case of scenario 2 or 3.
- `costDivider`: represents the denominator of the fraction we use to calculate what percentage of the total each data requester should pay in case of scenario 2 or 3.
- `mu = 0`: represents the default mean (μ) of the normal distribution we use to generate the data requesters' *action* probabilities.
- `sigma = 0.1`: represents the default standard deviation (σ) of the normal distribution we use to generate the data requesters' *action* probabilities.
- `scenario = 2`: is the default scenario.
- `accessTimePeriod = 2`: represents the default value of how many seconds requesters will have access to the data.
- `maxDataProviders = 1`: represents the default maximum number of data providers we allow in this simulation.
- `providerProbabilities = 0.05`: represents the default maximum probability of a new data provider publishing or updating their data.
- `updateMultiplier = 5`: represents the default multiplier that increases the chance of a data provider making an update to their data. We use this variable to represent the relatively higher probability of a data provider making an update than publishing in the first place.

5.4 Output

In order to adequately visualize our simulation results we measure the following values each time a user takes *action* (all costs are noted in Dollar):

- `scenario`: to identify the simulation in case we build a dataset from multiple different simulations.
- `fraction`: the fraction of `costMultiplier/costDivider` to identify the simulation in case we build a dataset from multiple different simulations.

- `period`: as representation of time passing.
- `action`: to differentiate between *actions* being taken.
- `accountNumber`: to identify which user made the *action*.
- `type`: whether it was a data provider or data requester.
- `contractNumber`: in case we simulated more than 1 data provider.
- `totalCost`: a running total of all arising costs, regardless of how or where they arise.
- `transactionCost`: the total cost of the transaction resulting from the user's *action*.
- `contractCost`: the running total of the respective contract.
- `currentExpectedCost`: the expected cost for a data requester before they make a transaction (*request access* or *renew access*).
- `nextExpectedCost`: the expected cost for a data requester after they make a transaction (*request access* or *renew access*). Note that this does not necessarily have to be equal to the `currentExpectedCost` of the next data requester in line, since an update could have happened in between.
- `providerEarnings`: a running total of the amount transmitted to the contract as payment (always 0 in case of scenario 1).
- `providerCost`: a running total of the costs arising from the provider taking *action* (i.e. publishing or updating their data).

These data points are compiled to one comprehensive dataframe.

5.5 Starting the simulation

The first action in each simulation instance is the first data provider publishing their dataset. Then the loop starts. Our simulation implementation offers the option for multiple data providers to decide to publish over time.

In each *period* we will check for each of the four possible *actions*:

- *Publish*: exactly 1 data provider has the chance to publish (denoted by their probability of taking *action*). Until they do publish, no other data provider will be able to publish. This represents the passage of time (*periods*) between different providers publishing their data.

- *Update*: each data provider with a published dataset has the chance to issue an update. Since we assume that a data provider, once they published their dataset, would be legally required to update it somewhat regularly, we increase the chance to update by a certain factor (see `updateMultiplier` in 5.3).
- *Request*: exactly 1 data requester has the chance to request access to a randomly determined dataset among those available. If this data requester does not request access, they will have the same chance to do so in the next iteration of the loop until such a time where they do 'make the decision' to request access. Only then will the next data requester in line have the chance to make a request. This simulates the potential time gap between different requesters making access requests.
- *Renew*: each data requester with an access token will have the chance to renew their access time to the data. In our simulation, we assume that requesters will only renew access time if it has expired since this is economical behavior. A data requester may not know precisely for how long they need access, thus it makes sense to add access time only when needed, especially since potential costs in scenarios 2 and 3 are likely to be lower with each passing *period*.

Time in this simulation algorithm is an ambiguous subject. On the one hand, we simulate the passage of time by assigning probabilities to users that might or might not take *action*. On the other hand, we attribute access times in real seconds to the tokens generated upon a successful request or access renewal. Since the simulation would be flawed if these two systems do not operate synchronously, we implemented a condition that disallows access time renewal until 2 *periods* after the requester's last *action*. This reflects the idea that a *period* is roughly equivalent to a week, thus each data requester would be able to renew their access to the data for two weeks.

5.6 Determining optimal parameters

In this section, we describe how we determined the parameter values that lead to the results shown in chapter 6.

The most pivotal variables (apart from the scenario itself) are the `actionTicker`, and the cost fraction data requesters must pay when making *access requests* or *renewing* their access time, denoted by `costMultiplier` and `costDivider`.

We simulated scenario 2 to determine the optimal values for these variables, since this is the most dependent on actions (and scenario 3 is basically a multiple of scenario 2). For these test simulations we chose the following parameters:

- Total *actions*: [50,500,1000]
- Fractions: [0.50, 0.05, 0.01]

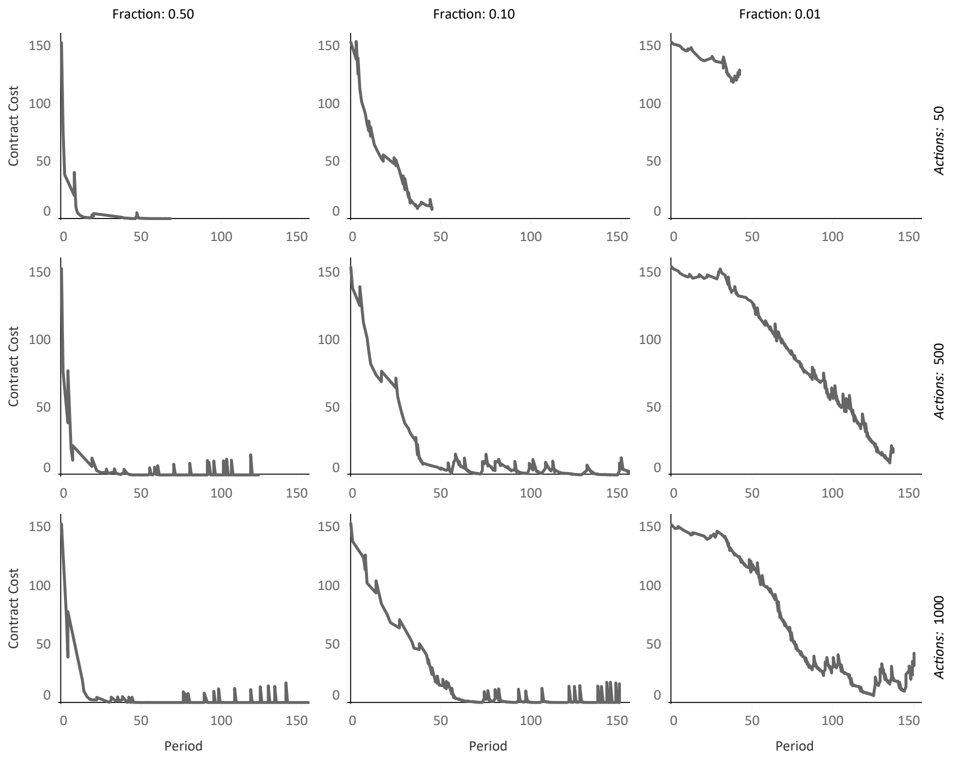


FIGURE 5.1: These graphs show the running total contract cost deducted by what data requester's pay in scenario 2.

In the resulting figure 5.1 we can observe that a high percentage cost distribution (i.e. what fraction a data requester must pay in return for access) leads to a too rapid decline in the running contract cost and immediate coverage of new arising costs whenever the data provider updates. While the data provider might be quite happy with this, it is inherently unfair to the data requesters, since some will pay high amounts while others pay almost nothing. On the other extreme, when data requesters pay only a small fraction of the running contract costs we can observe a balancing of revenue and expenses above zero, which is not the goal of scenario 2. Thus, we can safely conclude that the fraction must lie between the extremes to be effective.

The answer to one of the goals posited in the introduction to this chapter (5), how quickly the costs of the data provider are covered, and how well new arising costs from updates are covered after the initial deployment costs are covered can be deduced from figure 5.1. We basically asked how many periods are sufficient to show this, and the answer is it depends on the fraction of the running costs data requesters pay. As such, we must answer what a fair fraction would be, which we do in the paragraph above. As such, the optimal parameters for our simulations are, judging from the graphs in figure 5.1, 500 *actions* and 5% cost coverage.

The *profit margin* for scenario 3 is set to 200%, meaning the data provider's total earnings in this scenario are exactly double that of their costs (making for 100% pure profit after covering costs).

Chapter 6

Evaluation and Results

In this chapter, we show the results of the simulations described above and how they achieve the goals described in section 5. For this purpose, we differentiate three scenarios that show the different transposition of the incentives discussed in section 2.3 into our prototype and the simulation. In section 6.2, we will show how costs arise from using the system, how they are covered in scenarios 2 and 3, and then describe how the system balances prices for the data requesters with fairness considerations. The results indirectly demonstrate the overall applicability of our system and the improvements to the smart contract structure introduced to LUCE in this thesis. This, in turn, showcases how the system applies **monetary** and **cost** incentives to motivate data providers to use the platform, which inherently results in incentives for data requesters to use the platform as well (due to resulting **data-availability**, **data-variety**, and relative **inexpensiveness**).

6.1 Incentive Scenarios

With the simulation of three different scenarios with escalating pricing strategies, we seek to improve understanding of how exactly costs can be covered.

First, we will describe how the scenarios differ from each other:

Scenario 1. All parties cover their own costs resulting from function calls. The main purpose here is to serve as a baseline comparison for the other two scenarios. Relevant incentives for this scenario are **research** and **reputation**. Since publishing costs in this scenario are covered by the data provider, this could be viewed as a donation made to support research, as it is given without want or consideration, i.e. without compensation. A weaker form of scenario 1 may be one where the data provider's costs are completely or partially covered by a third party through subsidies (e.g. if an insurance company wishes to have a stake in taking an active role in supporting medical research). In this case, the data provider would be similar to a non-profit organization.

Scenario 2. The costs resulting from function calls will be paid back in stages by the data requesters. This scenario is based on the fact that system inherent costs are relatively low compared to what data acquisition costs currently amount to (Getz et al., 2015). This will be shown later in this chapter (table 6.1). In this scenario, the amount a data requester is asked to pay upon requesting access or renewing access is based on a fraction (e.g. 10%) of the total cost accumulated from the original deployment of the contract, publication of the dataset, and other smart contract operations that have a *gas* cost, such as updates. This results in a gradual decline in running costs for the provider, which represents a fairness consideration. This means early data requesters will pay relatively more than later data requesters since already transmitted fees are deducted from the running cost total denoted internally in the smart contract. Relevant incentives for this scenario are **research** since data providers stand to benefit passively from resulting research, and **costs**, as they can participate at no financial cost. The **reputation** incentive has weaker relevance in this scenario than in scenario 1 since the data provider is not making a donation. In this scenario, they are equivalent to a non-profit organization.

Scenario 3. Here, the data provider can set a desired *profit margin* they wish to attain. Similarly to the cost distribution fraction described in scenario 2, this is represented by a percentage value (e.g. 120%) of the total cost arising from operations in the smart contract. Each data requester will still pay a fraction of the total costs, but this fraction will be respectively higher (e.g. 20% higher than the original costs). By adjusting the profit margin, a data provider can freely control the resulting price for their dataset. There is no limit on how high or low this profit margin can be. However, we expect that this free market will self-regulate, i.e. prices will not be unreasonable. Profits generated in this scenario may be redistributed to data subjects, which ties into the **monetary** incentive explained in section 2.3. As such, relevant incentives for this scenario are **research** and **monetary**, though once more it is hardly unreasonable to also expect a positive **reputation** gain from sharing valuable data for research, even if the provider profits from it, and especially if the data subject can profit as well.

The reason we choose to simulate these specific scenarios is that they represent the most basic permutations of cost allocation for the data provider that reflect the incentive types introduced in section 2.3: (i) no compensation, (ii) cost compensation, and (iii) profit. As such, this thesis will provide an overview of how costs arise in the system and how they may be covered and establish critical baselines necessary to facilitate the exploration of more complex scenarios in future work.

6.2 Cost analysis

Transactions on the Ethereum network (in this case a test network) have a *gas* cost that is directly proportional to the internal operations of the respective function call in the smart contract. Importantly, not all operations cost the same amount of gas. Specifically, storing data in on the blockchain is relatively expensive, since it requires space on the hard disks of all nodes in the network. This is why different data types have different associated costs, which scales with their respective byte count. As such, storing large text strings is more expensive, while storing small numbers is usually less expensive. In short, the cost of writing to the blockchain scales with the size of the content. This is why the deployment cost of a new smart contract is generally quite high compared to transactions resulting from calling the functions of that smart contract.

The table below shows the base costs of the four core functions of LUCE[1]. These are the pure transaction costs resulting from calling the respective function, which basically equates to scenario 1. In scenario 2 and 3, the *request* and *renew* functions require additional funds to be transmitted with each function call. The Dollar price is based on the *gas* (72 Gwei) and Ether ($316.52) prices at the time of writing this thesis.

Action	*Gas* cost	*Dollar* cost
Deployment	6,560,749	$149.52
Publish	95,688	$2.18
Update	47,999	$1.09
Request	465,316	$10.60
Renew	37,510	$0.85

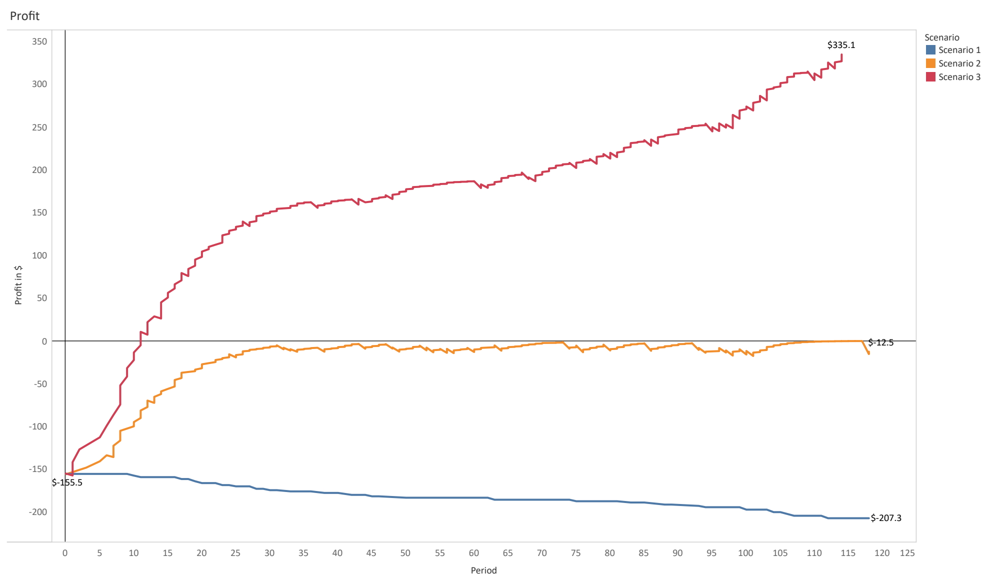
TABLE 6.1: Base cost for the core functions of LUCE

As mentioned before, the costs to update a dataset scale with its active users. This is why the cost is relatively low when there is no data requester ($1.09), but far higher when there are e.g. 60 data requesters ($29.60), which makes for roughly $0.47 per requester for an update. We will show that these comparatively higher costs are still easily covered by the system, both in the short term (figure 6.1) and in the long term (figure 6.6).

Figure 6.1 shows the profits generated in each scenario. We can see that after roughly 45 periods in scenario 2 costs are completely covered. As expected, in scenario 3 the break-even point is reached faster, and positive returns can be measured as soon as period 16. We can already roughly observe how updates impact the graphs (the spikes), and in scenario 1 the slow update cost upwards scaling in proportion to the number of data requesters is also visible.

As mentioned previously, the cost of updating the meta-information of the data in the smart contract scales with the number of requesters because each requester

[1]For a full table detailing all functions of the smart contracts and their inherent *gas* and Dollar costs, refer to Appendix B.



Period vs. Profit in $ and Profit in $ Scenario 3. Color shows details about Scenario. The marks are labeled by Profit in $ as an attribute. For pane Profit in $ Scenario 3: The marks are labeled by Profit in $ as an attribute and SUM([Profit in $ Scenario 3])/2. The view is filtered on Period, which includes values less than or equal to 118.0.

FIGURE 6.1: This visualization shows the total profit over time for each scenario.

must be notified of that update to give them a chance to comply. Figure 6.2 displays the relationship of running contract costs (grey line; the spikes are updates) and individual transactions in more detail. Here, we can more closely observe rising update cost (the blue X marks) and sinking access costs over time (the orange circles and plus signs). Each data requester in this scenario pays 5% of the running costs at the time of their request. With this setting, data providers in scenario 2 can veritably expect that their costs will always be covered under the condition that data requesters continue to use their dataset. Should this not be the case, e.g. if the dataset loses its value, cost coverage may take a longer time, or, in the extreme case, costs may not be covered. We discuss this problem in chapter 7. In our technical implementation, the only difference between scenario 2 and scenario 3 is the *profit margin*. Profits in scenario 3 are effectively a linear multiplication of costs in scenario 2 and follow the same arguments. However, since scenario 3 is explicitly profitable, it reaches the break-even point faster in proportion to how high the *profit margin* is set.

We can also observe the change in additional costs for data requesters. After initial deployment (periods 1-20), costs for requesters are higher than otherwise (periods after 20). In figure 6.2 there are 59 data requesters in total, simulated over 118 periods.

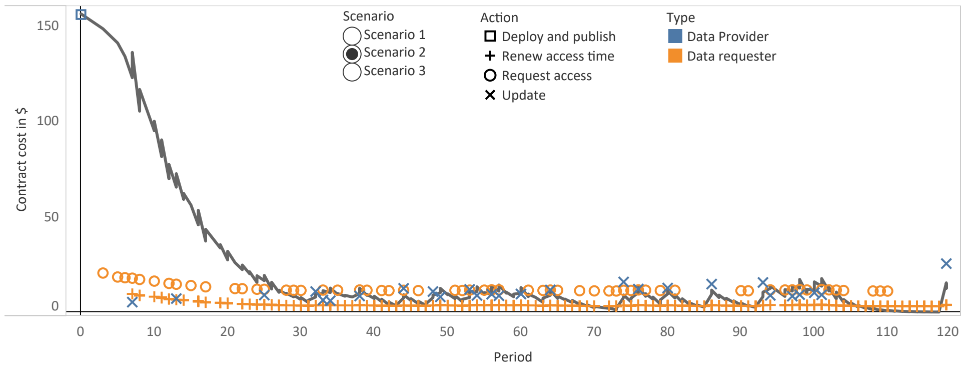


FIGURE 6.2: This graph shows how the running costs of a smart contract are influenced by individual transactions made by data provider and data requesters.

In total there are:

- 27 updates to the data (frequency 0.22/period)
- 59 access requests (frequency 0.48/period)
- 418 access time renewals (frequency 3.54/period)

This makes a total of 505 *actions* and reflects our assumption that there are far more data requesters than providers, and that data is very valuable.

The initial cost for a data requester is dependent on which scenario we are simulating. As mentioned in table 6.1, the base cost of requesting access is \$10.60. In the other two scenarios, a variable additional price is added to cover the data provider's cost or generate the data provider's profit respectively.

Figures 6.3 and 6.4 show requester costs specific to each scenario. We can observe the average base transaction cost for requester *action* types and the additional cost stacked on top (which the requester bears instead of the provider in case of scenario 2 or 3 respectively). Compared to what researchers currently pay for their data, these amounts may seem negligible, however, this notion is flawed since it disregards potentially rising Ether and *gas* prices. To put this into perspective, over the four months period of writing this thesis, the *gas* price for transactions has increased more than fourfold, and the price for Ether has increased by roughly forty percent[2].

[2]Price development from March 2020 to July 2020, all prices shown in this thesis are derived from the *gas* and Ether prices from 28.07.2020.

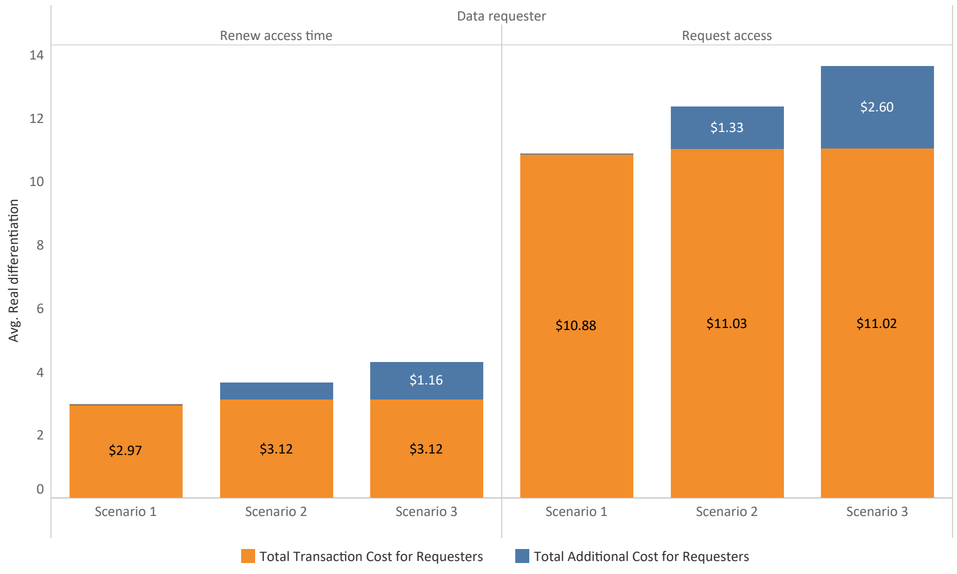


FIGURE 6.3: This bar chart shows the average base transaction cost and average additional transaction cost resulting data requesters must pay in order to access a dataset.

This clearly shows that prices can fluctuate heavily. Compared to requesters' individual costs, the data provider has much higher costs, as shown in figure 6.4. Over 118 periods, they must invest $220 and $290 (depends on random chance in our simulation). However, as demonstrated by our simulations (see figure 6.1), even the relatively high initial costs of deployment can be expected to be quickly recovered by the data provider in case of scenario 2 and 3. This reflects our assumption that there are far more data requesters than providers. If this were not the case, data providers would likely be forced to set higher cost allocation fractions to cover their costs (percentage of what each requester pays).

For a more detailed overview of what range of costs each user of the platform can expect, we plot the simulated cost distributions based on each *action* type in figure 6.5.

We can infer from figure 6.5 that there are quite a few outliers in regard to the cost distribution among data requesters, both when initially requesting access, and when renewing that access. This can be attributed to the fact that the first five to ten early requesters cover the majority of initial deployment costs, which are generally much higher than update costs. If we refer back to figure 5.1, we can observe how this

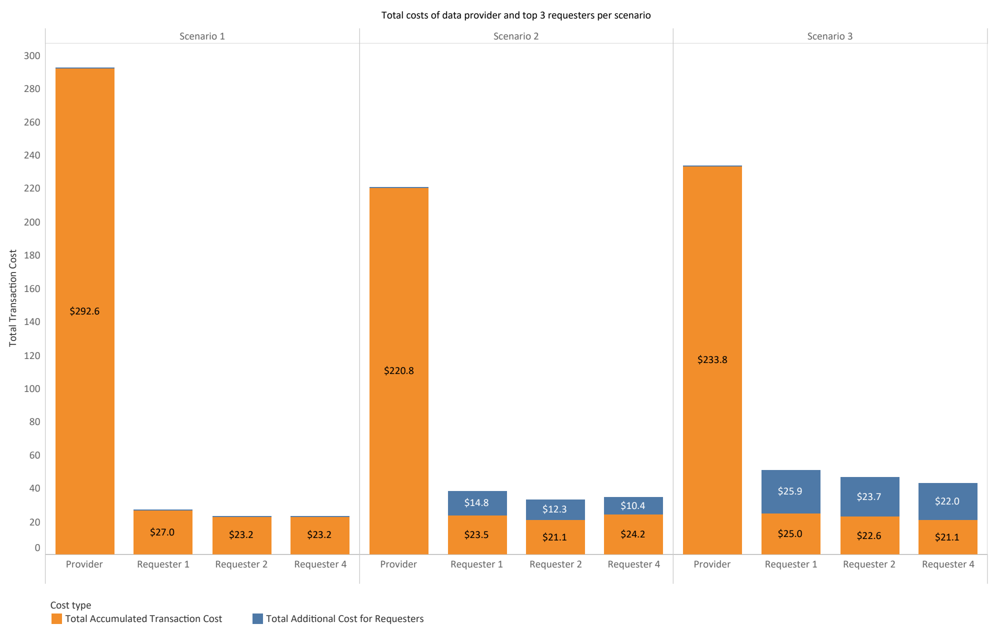


FIGURE 6.4: This bar chart shows the total cost sum of the data provider and the top three requesters per scenario and differentiates between base transaction costs and additional costs resulting from scenario 2 and 3.

unequal distribution of costs could be smoothed out by choosing a smaller fraction to denote what percentage of the running costs requesters must cover. However, if this fraction is chosen too small, it will likely lead to the data provider's cost not being covered, which defeats the purpose of scenario 2. This is why we determined a fraction equating to 5% cost coverage per data requester in the previous chapter to be optimal.

The simulations show that under the assumptions from section 5.1 regarding user behavior, the system behaves as predicted. Scenario 1 shows the cost development without any cost intervention, i.e. each user pays only their own expenses. Noticeably, the individual cost of each data requester is far outstripped by the running costs of the data provider. However, based on the assumption that several data requesters will seek access to any given dataset, scenarios 2 and 3 demonstrate that up-front costs paid by the data provider will be quickly recovered.

Interestingly, we can observe in figure 6.6 that even a very low percentage of running costs paid by requesters eventually balances with the cost of updates. However, this does not mean the initial deployment costs, which are typically far higher, will

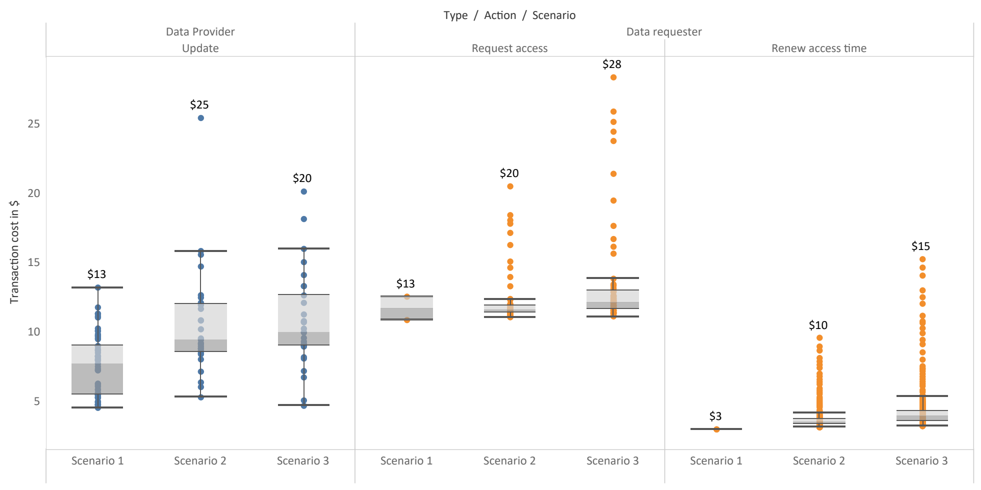


FIGURE 6.5: This boxplot shows the range of costs arising for each user type and each *action* type. The Dollar value notes the maximum amount for each category. Note that the y-axis of this plot is logarithmic.

also be covered. In fact, figure 6.6 clearly shows that an average of $49 of the initial investment are never recovered, which means in this case the system fails to deliver on the **cost** incentive connected to scenario 2.

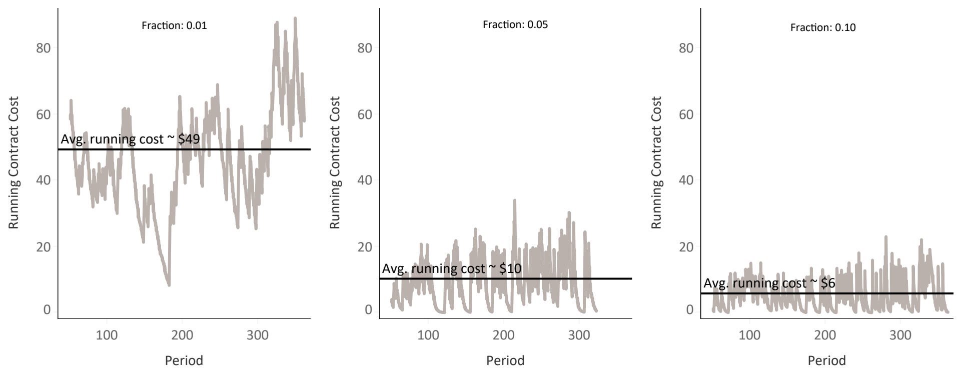


FIGURE 6.6: These graphs show the long-term sustainability of the system for 1%, 5%, and 10% cost coverage per requester.

The graphs in figure 6.6 represent simulations of scenario 2, with 2000 *actions* each. This is why the graphs extend to period 300 and beyond (if we follow the notion that 1 period equals 1 week, this would mean a simulation over more than 5 years). The left graph in figure 6.6 shows how updates influence the long term cost balance if each data requester pays only 1% of the running smart contract costs of the provider. In the middle and right graphs, we can see how different payment fractions change the cost coverage balance compared to the left graph in favor of the data provider. Additionally, we can observe how quickly the data provider's costs are covered according to each fraction by comparing the slopes of the graphs following each update. The left graph in figure 6.6 shows a far more gradual cost coverage following updates than the other two, where we can observe cost coverage taking place almost instantaneously. Naturally, the higher the fraction, the faster the cost coverage. However, if we infer the data requesters' average cost from these numbers, we can conclude that they are roughly the same ($0.49 in the left graph, $0.50 in the middle graph, $0.60 in the right graph). Thus, if our assumptions hold, it is reasonable to set the payment fraction somewhere between 5% and 10%, meaning requesters will not be charged unfairly and data providers will quickly recover their investments.

These results demonstrate the functionality of our extension to the LUCE smart contract structure and functions. With the simulations of the three distinct scenarios, we show that depending on the parameters set in the smart contract, (a) data providers face considerable up-front costs to cover the deployment of pertinent smart contracts and (b) that this initial investment, as well as all running costs incurred by the necessity to regularly update the data, can be rapidly recovered by data providers. Importantly, this suggests that both **cost** and **monetary** incentives would likely be effective means to motivate data providers to participate in the LUCE platform: In scenario 2, the ability for data providers to quickly recover up-front investments minimizes the disincentive that up-front costs would otherwise manifest. Consequently, the main positive incentives in scenario 2, **pertinent** and **reputation**, will likely not be significantly diminished by cost. Scenario 3 extends this by additionally introducing a **monetary** incentive. Here, costs incurred by data providers are covered with even stronger kinetics than in scenario 2, and they additionally benefit from profits, determined by the *profit margin* they set. Through our implementation of how payments by data requesters are calculated, data providers effectively cannot profit infinitely, depending on the frequency with which they make updates to their data. The profit calculation is directly derived from occurring costs. This means, if data requesters sufficiently outnumber data providers, there will come a point where the data provider has fully achieved their desired profit because it is a linear combination of their costs. From that point, only new costs incurred by the data provider, e.g. an update to their data, will result in new profit. This effectively limits how much a data provider can ever profit from high demand and since the same calculation is used for scenario 2, where no profit is generated, high demand will similarly result in costs being covered completely, which means requesters have no additional costs from requesting access to the data. In such cases of extremely high demand, it may be a valid fairness consideration of the data provider to lower the percentage of the running costs each data requester must pay. Conversely, if there is extremely low demand, the data provider may wish to increase this percentage.

As such, we provide the data provider the tools they need to control how their costs are covered or profits are generated.

Chapter 7

Discussion

The results in chapter 6 show that the improvements and advancements made on the original prototype of Angerer, 2019a successfully enable both **monetary** and **cost** incentives. Other incentives, such as **reputation** will require further analysis, though we do propose how reputation might be affected depending on the scenario, e.g. positive ramifications to the image of the participating institutions through quasi-donations (scenario 1), a status as non-profit organization in regards to participation on the platform (scenario 2), or simply open support for research initiatives through active data-sharing (all scenarios).

The simulation of scenario 1 shows how the costs of the data provider develop over time. If we assume that one period equals one week in real-time, then the data provider's yearly cost is roughly $170 per published dataset. Himmelstein et al., 2014 created an overview of hospital administrative costs, from which we can conclude that the cost of our system is negligible when considering the whole. This might change if Ethereum experiences strong growth. However, the emergence of Ethereum 2.0 (Buterin, 2017) in the future should counteract such a development to a certain degree.

It is also important to consider the **research** incentive. Since it is arguably in every human's best self-interest to support medical research, it is not unreasonable to expect some form of subsidy to the system, such as government or private funding, that facilitates the acceleration of medical innovation. However, LUCE is still in development and lacks several key components in regard to its final implementation, which we will discuss in section 8.1. As such, detailed analyses regarding subsidies and other financing models will need to be addressed in future work.

Our results show that in scenario 2 the costs of the data provider can be quickly recovered to 100%. An important question that remains is how long this would take in the real world. Since data providers must make initial investments to share their data, and scenario 2 promises to cover their costs, the time it takes to do so should not be unreasonably high. If we assume that one period equates to one week, as suggested earlier, then complete cost coverage would take roughly seven months. Conversely, if we assume that a period is a day, it would take less than one month to cover all costs. However, since all of this is based on rather stringent assumptions about the users of the system, it is impossible to deduce the number that reflects reality. The only way to reasonably predict this would be a study that surveys how data subjects, providers, and requesters would act if they had access to the system. We suggest one possible test environment for such an experiment in section 8.1. Nevertheless, given the low relative costs of data provision for the presumed participants (hospitals and major medical research institutions), even a conservative estimate of cost-recovery over several months would likely not present a significant disincentive for data providers.

In scenario 3 we observed that costs were covered rather quickly because we set the profit margin to 200%. Since scenario 3 is basically a linear multiplication of scenario 2, we can deduce how lower or higher profit margins would change the curve for scenario 3 in figure 6.1. Our simulation shows the effectiveness of the main **monetary** incentive associated with this scenario.

It also suggests the possibility of a mixture between scenarios 2 and 3. If the profit margin is set relatively low, e.g. 120%, then data providers would slowly build up a buffer that could be used to finance periods in which there are fewer data requesters. This could lead to a more stable system in practice. Additionally, this could also relieve the problem of some data becoming less valuable the more it is used: In particular, there will most likely come a point where some data sets become less and less useful for scientific research. At this time, it would be desirable to have built up a small buffer that could be used to cover the gas cost of finalizing the deletion of the dataset when it is no longer needed.

All three scenarios clearly show that the system functions as intended (within the limitations of our assumptions).

From a managerial perspective, the implementation of our model reveals different implications for each of the actors involved. The platform stands and falls with the volition of data subjects. This is why data ownership control and access exclusivity are requirements for the platform that cannot be circumvented. In that sense, we have brought the platform a significant step closer to realization. Data providers, on the other hand, must bear the initial costs of publishing data, irrespective of the scenario. As such, they are inherently disincentivized to use the platform, which we seek to alleviate with scenarios 2 and 3. As such, it is unlikely that data providers will participate in the platform if their resulting costs are not covered. However, this coverage can come from uninvolved parties as well, such as state subsidies or third-party investments. If we consider the data requester's perspective, all their requirements and incentives are almost automatically given by the existence of the system, if it can be realized as suggested in this thesis. This holds true even if they must partially cover the data provider's costs since we have seen in chapter 6 that their total costs for accessing a dataset over an extensive period of time rarely rise above $50. If we compare this to the current cost of data collection in the biomedical fields, the savings are very significant. However, to truly implement the system, there are still several important milestones ahead, the biggest of which is the creation of a supervisory authority that audits the system. This could be a government institution, or possibly a private audit firm. Additionally, we do not consider costs resulting from the ETL processes required to make data actually useful for analysis. Data providers would likely need to employ at least one person to facilitate the compilation of relevant medical data to be shared on our platform. These costs could be injected into the smart contract logic, meaning data requesters would ultimately defray these additional costs. However, if our assumption holds that data requesters far outnumber providers, this additional cost will likely not outstrip the costs outlined in this thesis by an insurmountable margin.

In terms of research implications of this thesis, we provide a baseline that clearly shows what system inherent costs can be expected from a blockchain-based data-sharing platform, and how these could be covered according to different user incentives. Specifically, our research contributes the following to the body of existing knowledge:

- A well-founded, intricate system of smart contracts that facilitate GDPR compliance of all parties involved, thus protects individual privacy.
- A basic implementation to facilitate and secure the registration of participants, to be controlled by an overarching institution with strongly isolated powers solely for verification and authentication.
- A sophisticated token generation system that facilitates the access barrier to the data shared on our platform. Our adaption of the ERC-721 token standard is applied to the specific use-case of data-sharing and implements several methods that differentiate it from other tokens, such as making it transferable only once, and only by the data provider.
- A unique cost control structure that allows data providers to precisely control how, how fast, and to what extent their costs are covered. We make use of the economic principle of profit margins by adapting the formula for cost calculation to be offset by a percentage that represents the combination of costs and desired profits. This way, we elegantly provide a single solution for both **cost** compensation and direct **monetary** rewards in the form of profits. The unique contribution of our system is how profits are effectively limited by the costs resulting from using the system (in our implementation profits are a linear combination of costs) instead of flat fees that always apply, and payments asked from requesters are dynamically adjusted according to the current running contract costs. The data provider is able to easily regulate this cost control structure by simply adjusting these two variables (profit margin and payment fraction).

As such, we open the pathways to practical test runs of the system in controlled environments. This could, for example, be gamification of the system at a university that allows students to take on the role of data requesters or data providers who can then optimize their occurring costs in return for some sort of reward. Such a system would most likely more closely represent a realistic simulation than what we created in this thesis since each participant would actively seek to optimize their respective utility.

In the following sections, we will discuss several limitations of the system and then compare the newest iteration of LUCE emerging from this thesis with the other two platforms, Ocean and MedRec, introduced in sections 2.4.1 and 2.4.2.

7.1 Limitations

7.1.1 Scalability

The issue of scalability generally applies to all blockchain applications. This means the system response times will slow down considerably as the number of transactions per unit of time grows. This limitation is especially impactful for cryptocurrencies that seek to become a replacement for, or at least an alternative to, payment systems that are currently commonplace (e.g. PayPal, VISA, MasterCard, and others, which process more than 5000 transactions per second and are capable of even more). In our system, this issue is secondary, since we do not expect transactions to be time-sensitive on the scale of seconds, but rather minutes. Transactions on the Ethereum network currently take roughly 15 seconds.

A different problem is the escalating cost of updates to a dataset's meta-information. This not only affects the potential data provider attracting a particularly large base of requesters, but it could also affect other users on the Ethereum blockchain, even those who have no stake in the LUCE platform. This is because all transactions are saved to the blockchain, which in turn is saved on all nodes of the Ethereum network. Thus, when storage space becomes a sparse commodity, *gas* prices will quickly rise to the point where transactions become cripplingly expensive. To put this into perspective, simulating one data provider and sixty data requesters, who take a total of 5000 *actions* resulted in roughly 2.5GB of data on the blockchain. At the time of writing this thesis, the Ethereum blockchain consists of 300GB of data. Ethereum 2.0 aims to somewhat alleviate this issue by storing only a fraction of the transaction data on each node. This will result in a much greater total capacity of the network (Buterin, 2017).

7.1.2 Registry

The idea behind the LUCE registry we propose in this thesis relies on a central institution that verifies users' credentials. This is not yet implemented. The registry smart contract we developed in this thesis grants authorization to any user of the system. This worked well for our simulations but is not compatible with any real-world testing, since it allows abuse. One way to implement a sustainable, scalable solution to this issue would be to rely on the expansion of other existing systems, such as the digital ID (DigiD, BKWI, 2003) in the Netherlands. Similar to our registry, DigiD does not store any data on individuals but merely associates a representative number with a true/false indicator of whether a person is registered or not. The LUCE registry could theoretically be integrated with this system and thus provide a secure identity validation service. Many other European and other countries offer similar options, and the eIDAS (European Union, 2014) regulation makes the provision that electronic identification methods must be accepted in all EU member states. This makes the integration of the LUCE registry that functions internationally quite feasible.

In addition to license and registration verification, we also experimented with adding entries for datasets in the same smart contract. This is a practical implementation of the *data catalog*, conceptualized by Havelange et al., 2019, to let data requesters browse the available datasets' meta-information. In the original prototype by Angerer, 2019a, the *data catalog* was stored on the LUCE web server, which opens it to potential security issues. However, due to technical limitations introduced by Jameson, 2016, we were unable to include this feature in our simulations. Tests on Remix IDE, 2020 were successful, so we are confident future versions of LUCE will implement it successfully. Whether or not it is necessary to duplicate all meta-information on an additional smart contract and thus further increase the cost of running the platform is debatable. The danger of storing the datasets' meta-information on less secure servers is related to how the system currently associates datasets with their corresponding smart contracts, i.e. the public address of that smart contract. If a person with malicious intent were to hack the server and change this public address to a fake contract of their own, they could effectively mislead data requesters into transferring them money which cannot be recovered.

The adaption of the meta-information storage suggested here is only one possible solution to this problem. A different solution could be a more direct communication of data requesters and providers or splitting meta-information that could introduce vulnerabilities to the system (i.e. contract addresses), and non-vital information (such as the data description), because this could be looked up manually once the contract address is secured.

7.1.3 Volatility of Ethereum

As mentioned in section 6.2, both the Dollar price of Ether and the *gas* price for operations is volatile. Specialized tokens such as DAI (Tran, 2020) seek to provide alternatives for the currency aspect of Ether, however, there is no such solution for *gas* costs arising from transactions, which can only be paid in Ether. Thus, the LUCE platform is inherently susceptible to price fluctuations of Ether. As mentioned in section 7.1.1 on scalability, this issue is compounded by the *gas* price increasing in proportion to how many operations are being performed on the Ethereum network (i.e. proportional to needed storage space).

This could lead to maintenance costs (updates) or access requests becoming extremely expensive. While Ethereum 2.0 promises to somewhat alleviate this problem, by implementing the PoS consensus mechanism and data storage distribution on nodes, it remains to be seen how effective it will be. In the end, the distribution of data among nodes instead of saving the whole blockchain on every node delays storage problem but does not eliminate it. As such, there is not yet a perfect solution to the cost volatility of the platform, and data requesters may need to pay far more in one period than in a different period.

7.1.4 Comparison to Ocean and MedRec

If we compare the incentive structure of LUCE implemented in this thesis to that of the Ocean protocol, several similarities and differences become apparent. As mentioned in section 2.4.2, the Ocean protocol offers providers the option to be monetarily rewarded for profit (**monetary** incentive). This is similar to how we built scenario

3 in our simulations. However, the main incentive of the system we propose relates specifically to the parties involved. While Ocean seeks to establish data-markets that anyone can partake in and profit off, LUCE places several restrictions on both data providers and requesters. As such, data providers will be institutions such as hospitals that directly represent the wishes of their data subjects (i.e. patients) in regard to sharing their data, and data requesters are limited to academic researchers. This means the basic principle of operation of LUCE is to expedite research that will result in an improvement of worldwide welfare. The Ocean protocol concentrates on establishing data-sharing relationships between companies that do not necessarily represent the wishes of their data subjects. This will potentially have repercussions on other incentive types, such as **reputation**.

The main incentive of MedRec is to provide patients an accessible medical history. As such, it is closer to a blockchain-based EHR system than a data-sharing platform. This system is powered by the incentive of data rewards in return for mining transactions on the blockchain. This way, MedRec theoretically supports medical research. It is theoretical because on a public blockchain there is no absolute guarantee that a miner is indeed a medical researcher. If their smart contracts are, similar to LUCE, hosted on the Ethereum network, MedRec cannot specifically control who mines the specific block related to their transactions. Even if MedRec, as according to Ekblaw et al., 2016, implements a feature that allows miners to target specific datasets as mining rewards, this does not mean other miners are excluded. They discuss circumventing this issue by creating their own, private, permissioned blockchain. This would mean they lose the aspect of decentralization, transparency, and the trustless environment Partz, 2019. They posit the purpose of this change is geared towards privacy regarding the patients as record holders on the blockchain. Miners on this private blockchain would be permissioned medical researchers, who will then receive patient records as mining rewards. This system functions similarly to how LUCE only allows registered users to access datasets. In contrast to LUCE, a privately hosted blockchain would effectively limit the data control, and ownership claims data subjects can truly make because a centralized system does not offer the same trustless environment and transparency as a public one. Regarding the incentive system, MedRec's idea is quite innovative, since it partially circumvents the issue of gas costs and inherently supplies additional data storage by incentivizing researchers to participate in the mining process. However, this is counteracted by the notion of storing medical records on the blockchain, even if these are just pointers. As such, it is likely that the system is not scalable, since patient records are frequently made, updated, etc., and will thus require a large amount of storage, which might ultimately result in even greater **costs** for the data subjects.

Chapter 8

Conclusion and future work

8.1 Future Work

Though we made considerable advancements in this thesis, to create a practically feasible system for data-sharing, including the necessary incentives for data subjects and data providers, a number of open questions remain for future research and development:

The most prominent feature that remains purely conceptual is the power to be given to the supervisory authority to enable sufficient auditing possibilities. The only steps we've taken in this direction are the events and public transactions that are logged to the blockchain. However, there is not yet any system in place that allows the supervising authority to easily and reliably gather that data.

Similarly, there is no specialized way for data subjects to access relevant parts of the platform. Though much of the meta-information is publicly available, data subjects do not yet have a way to unequivocally identify which datasets hold their personal (anonymized) data. Though we do conceptualize a solution to this via hashing of the datasets, this is not yet implemented.

An institutionalized approach for registration control, and identity, license, and potentially data quality verification is still missing, even if the groundwork for this already exists. There is currently no controlling element in place that could handle this. Since licenses would be directly associated with the individual's public key, their wallet will become synonymous with their identity, which could in turn theoretically also interface with other systems for which crypto-wallets are needed.

The implementation of secure data storage is another unresolved issue, which will be tackled in future iterations of the platform. Dubovitskaya et al., 2015 propose an architecture that could be used to encrypt and store data on a cloud platform. This could be implemented on an existing data storage service such as Amazon S3 cloud.

Additionally, once this infrastructure exists, it can be tied in with the smart contract logic introduced in this thesis. Specifically, when a data requester downloads their requested dataset from the secure cloud storage, the system could automatically trigger their compliance function in the respective smart contract. Though this would be a soft mechanism, it would provide a legal basis for data requesters and the supervisory authority due to the transaction being written to the blockchain.

To improve scalability and especially the speed of the platform, we suggest considering the use of checksums in the secure storage system to allow users to view data without having to call up information from the blockchain. This could be especially useful for parties who have no direct stake in the system, such as data subjects and the supervisory authority.

Xuan et al., 2020 offer a highly mathematical analysis of participation strategies in data-sharing blockchain applications. They derive four conditions for which they model user participation in the system and create an incentive method that results in a stable user base, i.e. no over or undersaturation of users willing to share data. This could provide a basis for a more sophisticated simulation that derives participation probabilities from gain functions and pricing strategies, instead of basing them on a static normal distribution.

The next steps in the development of a sustainable LUCE platform applicable in the real world are to integrate the consent mechanisms created by Jaiman and Urovi, 2020, and a test application in a closed environment (such as the campus at Maastricht University) with real participants so we can observe how data providers and requesters behave in the system and what improvements need to be made to the platform. In this case, testing would most likely require a small private blockchain, so that users do not need to invest any real money into the system, but can experiment with artificial Ether to test the smart contracts underlying LUCE. Still, some form of reward to motivate actual participation would likely be required for a successful test. This could be a system where ’providers’ are awarded research tokens whenever a ’requester’ uses their ’data’, which could then be used as coupons for free meals in the cafeteria or similar incentives. This way, we could test how well the new system performs according to the measures established by Davis and Venkatesh, 2000 in their technology acceptance model.

8.2 Conclusion

With this thesis, we achieved our goal of conceptualizing and exploring different incentive types to motivate data providers to use the platform to share their data. We established the boundary conditions of data-sharing and derived requirements and incentives for the platform from them. We implemented the necessary solutions to the data-sharing challenges, by extending the smart contracts in terms of data ownership control, access exclusivity, internal logic, as well as cost control structures. The cost control, in particular, allows data providers to control the settings of how they wish to deal with arising costs, which is in line with **monetary** and **cost** incentives. We simulated the result for each option and provided a detailed breakdown of the costs inherent to using the platform for both data providers and requesters.

With respect to the LUCE project as a whole, this thesis provides several important advancements towards the real-world feasibility of the platform. We completely revised the trivialized smart contract logic of Angerer, 2019a, thus tackled one of the most important milestones of the prototype. These improvements include far better security, implementation of time-limited, sophisticated access tokens, and better integration with the GDPR from the side of dataset smart contracts. For the authorization and verification for all users of the platform, we implemented the first steps towards an institutionalized approach for handling the registration of users.

The simulated cost resulting from each of the scenarios compares favorably to those of current data-acquisition methods. From MRC, 2020 we can see that the budget for research grants by the Medical Research Council in the UK is capped at £1M. If we assume that typically half of that budget is used for data acquisition, it becomes quickly apparent that our solution could not only accelerate research, but also allow more, and more diverse research simultaneously. If we take this further and compare the cost of our system to business deals such as the one between Genentech and 23andMe, the cost disparity becomes even more apparent.

Our simulations have shown that the system prompts unavoidable costs and these must be financed. The three scenarios simulated in this thesis represent possible ways to tackle this issue, but others could also be viable. An example of this might be subsidies from the state or even companies with a staked interest in the platform's success. Either way, with this thesis we have provided detailed information as to the costs arising from sharing data via LUCE, suggested, explored, and analyzed how different incentives might motivate users, especially data providers to use the platform and connected these to three distinct scenarios that can be used as reference points for future research. What scenario or what incentive will lead to the best results when the platform goes into a non-artificial testing environment remains to be seen; however, we think this thesis has provided a solid foundation on which these questions can be explored and finally answered.

Shrestha and Vassileva, 2019 introduce a basic functioning framework for data-sharing via blockchain authentication similar to what LUCE envisions. Apart from the system inherent incentives that result from sharing research data, they focus on a monetary compensation incentive for data providers. However, they do not go into detail regarding what payments of requesters to providers should amount to, i.e. whether profit is generated or the system strives to achieve break-even. With this thesis, we contribute a detailed perspective of costs resulting from data-sharing platforms utilizing a comprehensive, extended, and easily reproducible prototype with sophisticated smart contract logic. With this, we show how users can be incentivized to participate in the platform, and what ramifications different cost allocations result in.

Finally, we can answer the research question of this thesis: What incentives will motivate users to interact with the LUCE platform and how can these be realized and supported through technology? We have established data-sharing incentive types according to economic, social, and health perspectives and derived three distinct scenarios from them. With these scenarios, we were able to test and simulate how specifically **monetary** incentives impact the data-sharing platform, given the necessary technical implementations. Societal (reputation-based) and health (research) incentives were harder to quantify and require their own, detailed studies. However, with this thesis, we have extended the technological and theoretical basis necessary to perform these studies in future versions of the LUCE platform.

Appendix A

Smart Contracts

- GitHub repository: `https://github.com/Leonard-Pernice/LUCE/tree/LUCE_develop`
- Main LUCE smart contract: `https://github.com/Leonard-Pernice/LUCE/blob/LUCE_develop/luce_vm/jupyter/data/new_luce.sol`
- Expanded LUCE registry: `https://github.com/Leonard-Pernice/LUCE/blob/LUCE_develop/luce_vm/jupyter/data/LUCE_registry.sol`
- Jupyter Notebook for simulations: `https://github.com/Leonard-Pernice/LUCE/blob/LUCE_develop/luce_vm/jupyter/x)%20Leon%20Simulations.ipynb`

Appendix B

Smart Contract Function Cost

TABLE B.1: Cost listing of all functions of the LUCE Registry.

Action	Transaction *gas* cost	Execution *gas* cost	*Ether* cost	*Dollar* cost
Deployment	621087	432315	0.04472	$14.15
newDataProvider	44855	22175	0.00323	$1.02
registerNewUser	45669	22797	0.00329	$1.04
updateUserLicense	27732	6268	0.00200	$0.63
checkProvider	23991	1311	0.00173	$0.55
checkUser	23877	1197	0.00172	$0.54

TABLE B.2: Base cost for the core functions of LUCE

Action	Transaction *gas* cost	Execution *gas* cost	*Ether* cost	*Dollar* cost
Deployment	6724230	5118378	0.48414	$153.24
publishData	95560	72560	0.00688	$2.18
updateData	43799	20863	0.00315	$1.00
addDataRequester	475067	453411	0.03420	$10.83
renewToken	45211	23747	0.00326	$1.03
setLicense	39339	37075	0.00283	$0.90
confirmCompliance	24218	2946	0.00174	$0.55
burn	65965	110465	0.00475	$1.50
setRegistryAddress	37131	14515	0.00267	$0.85
setScenario	27710	6246	0.00200	$0.63
setProfitMargin	35091	13627	0.00253	$0.80
setMultis	36148	14492	0.00260	$0.82
setPrice	31062	9406	0.00224	$0.71
receiveFunds	37223	16015	0.00268	$0.85
destroy	14106	6940	0.00102	$0.32
contractBalance	22469	1197	0.00162	$0.51
currentCost	22411	1139	0.00161	$0.51
getAccessTime	26711	5247	0.00192	$0.61
getAllDataRequesters	27123	5851	0.00195	$0.62
getCompliance	24944	2264	0.00180	$0.57
getLicense	22353	1081	0.00161	$0.51
getLink	25718	4254	0.00185	$0.59
getTokenId	25800	3120	0.00186	$0.59
userOf	24732	3268	0.00178	$0.56

Appendix C

Additional Figures

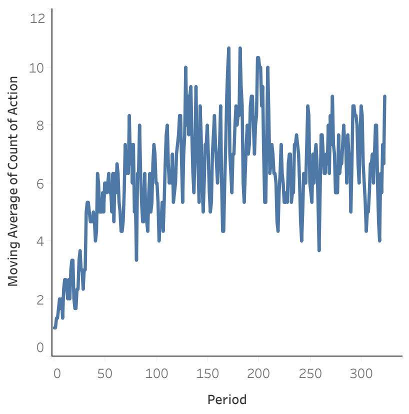


FIGURE C.1: This graph shows how the average actions taken per period eventually balance, as some requesters no longer renew their access and new requesters start doing so.

Bibliography

23andMe.com (2020). *23andMe Services*. URL: https://www.23andme.com/compare-dna-tests/?mkbanner=true. (accessed: 19.07.2020).

Angerer, A. (2019a). "A blockchain based architecture and implementation of the License accoUntability and CompliancE (LUCE) Data Sharing Framework". In: *Master Thesis, Maastricht University*.

— (2019b). *LUCE Technical Prototype and LuceVM*. URL: https://github.com/arnoan/LUCE. (accessed: 11.08.2020).

Azaria, A. et al. (2016). "MedRec: Using Blockchain for Medical Data Access and Permission Management". In: *2016 2nd International Conference on Open and Big Data*. DOI: 10.1109/OBD.2016.11.

Becze, M. and H. Jameson (2020). *EIP*. URL: https://eips.ethereum.org/all. (accessed: 24.07.2020).

BKWI (2003). *DigiD*. URL: https://www.digid.nl/en/login-methods/id-check. (accessed: 30.07.2020).

Bourke, A. and G. Bourke (2020). *Who owns patient data? The answer is not that simple*. URL: https://blogs.bmj.com/bmj/2020/08/06/who-owns-patient-data-the-answer-is-not-that-simple/. (accessed: 08.08.2020).

Buterin, V. (2013). "A Next Generation Smart Contract and Decentralized Application Platform". In: *White Paper*. URL: https://ethereum.org/en/whitepaper/.

— (2017). "Ethereum 2.0 Mauve Paper". In: *Mauve paper*. URL: https://cdn.hackaday.io/files/10879465447136/Mauve\%20Paper\%20Vitalik.pdf.

Catania, L., S. Grassi, and F. Ravazzolo (2018). "Predicting the Volatility of Cryptocurrency Time-Series". In: *Mathematical and Statistical Methods for Actuarial Sciences and Finance*. URL: https://doi.org/10.1007/978-3-319-89824-7_37.

Chez, B. (2020). *CoinMarketCap*. URL: https://coinmarketcap.com/currencies/ethereum/. (accessed: 28.07.2020).

chriseth (2020a). *In depth Contracts*. URL: https://solidity.readthedocs.io/en/v0.6.4/contracts.html. (accessed: 19.07.2020).

— (2020b). *Introduction to Smart Contracts*. URL: https://solidity.readthedocs.io/en/latest/introduction-to-smart-contracts.html#blockchain-basics. (accessed: 19.07.2020).

cipher.com (2019). *51% Rule*. How Blockchain Can Be Hacked: The 51% Rule and More. URL: https://cipher.com/blog/how-blockchain-can-be-hacked-the-51-rule-and-more/. (accessed: 07.07.2020).

Davis, F. D., R. P. Bagozzi, and P. R. Warshaw (1989). "User Acceptance of Computer Technology: A Comparison of Two Theoretical Models". In: *Management Science Journal* 35.8. DOI: https://doi.org/10.1287/mnsc.35.8.982.

Davis, F. D. and V. Venkatesh (2000). "A Theoretical Extension of the Technology Acceptance Model: Four Longitudinal Field Studies". In: *Management Science Journal* 46.2. DOI: https://doi.org/10.1287/mnsc.46.2.186.11926.

Davis, I.A. and L.J. Ryals (2010). "The Role of Social Capital in the Success of Fair Trade". In: *Journal of Business Ethics* 96. URL: https://doi.org/10.1007/s10551-010-0468-3.

De Angelis, S. et al. (2017). "PBFT vs Proof-of-Authority: Applying the CAP Theorem to Permissioned Blockchain". In: DOI: http://ceur-ws.org/Vol-2058/paper-06.pdf.

Digiconomist (2020). *Ethereum Energy Cost*. URL: https://digiconomist.net/ethereum-energy-consumption. (accessed: 19.07.2020).

Dubovitskaya, A. et al. (2015). "A Cloud-Based eHealth Architecture for Privacy Preserving Data Integration". In: DOI: https://doi.org/10.1007/978-3-319-18467-8_39.

Duch-Brown, N., B. Martens, and F. Mueller-Langer (2017). "The Economics of Ownership, Access and Trade in Digital Data". In: URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2914144.

Dyke, S.O.M. et al. (2016). "Consent Codes: Upholding Standard Data Use Conditions". In: *PLOS Genetics*. DOI: 10.1371/journal.pgen.1005772.

Ekblaw, A. et al. (2016). "MedRec: Using Blockchain for Medical Data Access and Permission Management". In: *2016 2nd International Conference on Open and Big Data*. DOI: 10.1109/OBD.2016.11.

Entriken, W. et al. (2018). *ERC-721*. URL: https://eips.ethereum.org/EIPS/eip-721. (accessed: 23.07.2020).

Erlich, Y., I. Pe'er, and S. Carmi (2018). "Identity inference of genomic data using long-range familial searches". In: *Science* 362 (6415). DOI: 10.1126/science.aau4832.

Ethgasstation (2020). *EthGasStation*. URL: https://ethgasstation.info/calculatorTxV.php. (accessed: 28.07.2020).

European Union (2014). *eIDAS*. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv\%3AOJ.L_.2014.257.01.0073.01.ENG. (accessed: 04.08.2020).

— (2016). *GDPR*. General Data Protection Regulation. URL: https://gdpr-info.eu/. (accessed: 07.07.2020).

Fecher, B. et al. (2015). "A Reputation Economy: Results from an Empirical Survey on Academic Data Sharing". In: URL: https://arxiv.org/ftp/arxiv/papers/1503/1503.00481.pdf.

Ganache (2020). *Ganache*. URL: https://www.trufflesuite.com/docs/ganache/overview. (accessed: 27.07.2020).

Gervais, A. et al. (2016). "On the Security and Performance of Proof of Work Blockchains". In: *CCS '16: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. DOI: https://doi.org/10.1145/2976749.2978341.

Getz, K.A. et al. (2015). "Quantifying the Magnitude and Cost of Collecting Extraneous Protocol Data". In: *American Journal of Therapeutics* 22. DOI: 10.1097/MJT.0b013e31826fc4aa.

Green, R.C. et al. (2013). "ACMG recommendations for reporting of incidental findings in clinical exome and genome sequencing". In: *Genetics in Medicine* 15.7. DOI: https://www.nature.com/articles/gim201373..

Grier, S.A. and S. Kumanyika (2010). "Targeted Marketing and Public Health". In: *Annual Review of Public Health* 31. DOI: 10.1146/annurev.publhealth.012809.103607.

Havelange, A. et al. (Aug. 2019). "LUCE: A Blockchain Solution for monitoring data License accoUntability and CompliancE". In: *Computing Research Repository*. URL: https://arxiv.org/pdf/1908.02287.pdf.

Hazari, S.S. (2019). "Design and Development of a Parallel Proof of Work for Permissionless Blockchain Systems". In: URL: https://ir.library.dc-uoit.ca/bitstream/10155/1037/1/Hazari_Shihab_Shahriar.pdf.

He, Y. et al. (2018). "A Blockchain Based Truthful Incentive Mechanism for Distributed P2P Applications". In: URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8329429.

Himmelstein, D.U. et al. (2014). "A Comparison Of Hospital Administrative Costs In Eight Nations: US Costs Exceed All Others By Far". In: *Health Affairs* 33.9. URL: https://www.healthaffairs.org/doi/pdf/10.1377/hlthaff.2013.1327.

Jaiman, V. and V. Urovi (2020). "A Consent Model for Blockchain-based Distributed Data Sharing Platforms". In: *IEEE Access*. DOI: 10.1109/ACCESS.2020.3014565.

Jameson, H. (2016). *Spurious Dragon*. Hard Fork No. 4: Spurious Dragon. URL: https://blog.ethereum.org/2016/11/18/hard-fork-no-4-spurious-dragon/#:~:text=This\%20update\%20prevents\%20an\%20attack,than\%20any\%20currently\%20deployed\%20contract.. (accessed: 27.07.2020).

Johnson, J.P. (2013). "Targeted advertising and advertising avoidance". In: *The RAND Journal of Economics* 44 (1). DOI: https://doi.org/10.1111/1756-2171.12014.

Lo, B. and D.L. DeMets (2016). "Incentives for Clinical Trialists to Share Data". In: *The New England Journal of Medicine* 375.12. URL: https://www.nejm.org/doi/pdf/10.1056/NEJMp1608351?articleTools=true.

Luu, L. et al. (2016). "Making Smart Contracts Smarter". In: *CCS '16: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. DOI: https://doi.org/10.1145/2976749.2978309.

Marani, H. et al. (2019). "Quality gaps of electronic health records (EHRs) in diabetes care". In: *Canadian Journal of Diabetes*. DOI: https://doi.org/10.1016/j.jcjd.2019.10.011.

Massessi, D. (2018). *Private/Public Keys*. Blockchain Public / Private Key Cryptography In A Nutshell. URL: https://medium.com/coinmonks/blockchain-public-private-key-cryptography-in-a-nutshell-b7776e475e7c#:~:

text=Public\%20keys\%20are\%20widely\%20distributed,can\%20decrypt\%20and\%20read\%20it.\&text=Blockchain\%20makes\%20extensive\%20use\%20of\%20public\%20key\%20cryptography.. (accessed: 24.07.2020).

McGowan, J.J., C.M. Cusack, and E.G. Poon (2008). "Formative Evaluation: A Critical Component in EHR Implementation". In: *Journal of the American Medical Informatics Association* 15 (3). DOI: https://doi.org/10.1197/jamia.M2584.

Molteni, M. (2018). *23andMe deal*. 23andMe's Pharma Deals Have Been the Plan All Along. URL: https://www.wired.com/story/23andme-glaxosmithkline-pharma-deal/. (accessed: 07.07.2020).

MRC, UK (2020). *Research Grant*. URL: https://mrc.ukri.org/funding/how-we-fund-research/research-grant/. (accessed: 19.07.2020).

Nakamoto, S. (2008). "Bitcoin: A Peer-to-Peer Electronic Cash System". In: *White paper*. URL: https://bitcoin.org/bitcoin.pdf.

Ocean Protocol Foundation (2019). "Ocean: A decentralized Data Exchange Protocol to Unlock Data for Artificial Intelligence". In: DOI: https://oceanprotocol.com/marketplace-framework.pdf.

Ocean Protocol Foundation, BigchainDB GmbH, and Newton Circus (2019). "Ocean Protocol: A Decentralized Substrate for AI Data and Services Technical Whitepaper". In: *White Paper*. DOI: https://oceanprotocol.com/tech-whitepaper.pdf.

OpenZeppelin (2020). *Sending Gasless Transactions*. URL: https://docs.openzeppelin.com/learn/sending-gasless-transactions. (accessed: 23.07.2020).

Partz, H. (2019). *Private, Enterprise Blockchains 'Make No Sense' and Are Set to Fail: Abra CEO*. URL: https://cointelegraph.com/news/private-enterprise-blockchains-make-no-sense-and-are-set-to-fail-abra-ceo. (accessed: 16.08.2020).

Remix IDE (2020). *Remix IDE*. URL: http://remix.ethereum.org. (accessed: 24.07.2020).

Ronquillo, J. G. et al. (2018). "Health IT, hacking and cybersecurity: national trends in data breaches of protected health information". In: *JAMIA Open* 1. URL: https://doi.org/10.1093/jamiaopen/ooy019.

Rosic, A. (2016). *Smart Contracts*. Smart Contracts: The Blockchain Technology That Will Replace Lawyers. URL: https://blockgeeks.com/guides/smart-contracts/. (accessed: 20.07.2020).

Rosic, A. (2018). *What is Ethereum Gas?* URL: https://blockgeeks.com/guides/ethereum-gas/. (accessed: 07.07.2020).

Sajana, P., M. Sindhu, and M. Sethumadhavan (2017). "On Blockchain Applications: Hyperledger Fabric And Ethereum". In: URL: http://www.smallake.kr/wp-content/uploads/2017/07/2017_Comparison-of-Ethereum-Hyperledger-Corda.pdf.

Shetty, S., C.A. Kamhoua, and L.L. Njilla (2019). *Blockchain for Distributed Systems Security*. AN ESSENTIAL GUIDE TO USING BLOCKCHAIN TO PROVIDE FLEXIBILITY, COST-SAVINGS, AND SECURITY TO DATA MANAGEMENT, DATA ANALYSIS, AND INFORMATION SHARING. John Wiley and Sons. URL: https://books.google.nl/books?id=c0aLDwAAQBAJ.

Shrestha, A.K. and J. Vassileva (2019). "User Data Sharing Frameworks: A Blockchain-Based Incentive Solution". In: URL: https://arxiv.org/ftp/arxiv/papers/1910/1910.11927.pdf.

Singh, N. (2020). *Permissioned vs Permissionless Blockchains*. URL: https://101blockchains.com/permissioned-vs-permissionless-blockchains/. (accessed: 20.07.2020).

Tenopir, C. et al. (2011). "Sharing Data: Practices, Barriers, and Incentives". In: URL: https://asistdl.onlinelibrary.wiley.com/doi/pdf/10.1002/meet.2011.14504801026.

Tran, K.C. (2020). *What is Dai?* URL: https://decrypt.co/resources/dai-explained-guide-ethereum-stablecoin. (accessed: 24.07.2020).

Valenta, M. and P. Sandner (2018). "Comparison of Ethereum, Hyperledger Fabric and Corda". In: *International Journal of Pure and Applied Mathematics* 118.18. URL: https://acadpubl.eu/jsi/2018-118-18/articles/18c/84.pdf.

Vogelsteller, F. and V. Buterin (2015). *ERC-20*. URL: https://eips.ethereum.org/EIPS/eip-20. (accessed: 23.07.2020).

Walker, R. M. et al. (2019). "Local health department adoption of electronic health records and health information exchanges and its impact on population health". In: *Collegian*. DOI: https://doi.org/10.1016/j.colegn.2019.06.006.

Wu, B. (2019). *Ethereum Scalability*. Ethereum Blockchain Performance and Scalability. URL: https://blog.bybit.com/academy/blockchain/ethereum-blockchain-performance-and-scalability/#:~:text=Like\%20Bitcoin\%2C\%20the\

%20main\%20reason,has\%20to\%20process\%20each\%20transaction.\&text=This\%20greatly\%20limits\%20the\%20transaction,of\%20the\%20Ethereum\%20blockchain\%20network.. (accessed: 19.07.2020).

Xuan, S. et al. (2020). "An incentive mechanism for data sharing based on blockchain with smart contracts". In: *Computers and Electrical Engineering* 83. DOI: https://doi.org/10.1016/j.compeleceng.2020.106587.

Yeung, T. (2019). "Local health department adoption of electronic health records and health information exchanges and its impact on population health". In: *International Journal of Medical Informatics* 128, pp. 1–6.