TU WIEN Informatics

Reducing the operation cost of a file fixity storage service on the Ethereum blockchain by utilizing pool testing strategies.

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Data Science

eingereicht von

Bsc. Michael Etschbacher
Matrikelnummer 51828999

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.univ.Prof. Dr. Andreas Rauber

Wien, 19. Mai 2022

Michael Etschbacher	Andreas Rauber

TU WIEN Informatics

Reducing the operation cost of a file fixity storage service on the Ethereum blockchain by utilizing pool testing strategies.

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Data Science

by

Bsc. Michael Etschbacher
Registration Number 51828999

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.univ.Prof. Dr. Andreas Rauber

Vienna, 19th May, 2022

Michael Etschbacher	Andreas Rauber

Declaration of Authorship

Bsc. Michael Etschbacher

I hereby declare that I have written this Doctoral Thesis independently, that I have completely specified the utilized sources and resources and that I have definitely marked all parts of the work - including tables, maps and figures - which belong to other works or to the internet, literally or extracted, by referencing the source as borrowed.

Vienna, 19th May, 2022

Michael Etschbacher

Kurzfassung

In dieser Arbeit geht es darum die Datenintegrität eines Digitales Archives, in welchem Kulturgüter wie Bilder oder Texte archiviert sind, mit Hilfe von blockchain Technologie zu sichern. Ich verschaffe einen Überblick über digitale Archive und das Konzept von Vertrauen, Authentizität und Integrität von Daten in Archiven und eine Methode, um diese Konzepte zu stärken. In der modernen Zeit, in der das Fälschen und Manipulieren von Daten einfacher denn je ist, ist es umso wichtiger, das Vertrauen in öffentliche Einrichtungen wie Archive zu stärken. Um die Datenintegrität zu gewährleisten, werden üblicherweise Kryptografische Verfahren angewandt, um sogenannte Fixitäts Informationen zu erstellen. Fixität ist ein Attribut eines digitalen Objekts, mit welchem validiert werden kann, dass ein Objekt über einen gewissen Zeitraum nicht manipuliert wurde. Auch wenn das Erstellen von Fixitäts Informationen (z.B. MD5, SHA256) relativ einfach ist, ist das Speichern von dessen umso schwerer, wenn Sie bedenken, dass jeder der diese ändern kann auch die zu Grunde liegenden Daten ändern kann. Als Speichermedium für die Fixität stelle Ich in dieser Arbeit die Ethereum blockchain vor. Es wurde bereits gezeigt, dass die Ethereum blockchain geeignet ist, um Metadaten unveränderbar zu persistieren, jedoch sind die Kosten für eine Individuelle Strategie, in welcher die Metadaten jedes Objektes einzeln in der Blockchain persistiert werden, immens. Um den Kosten entgegenzuwirken, werde Ich eine Pool-Testing-Strategie anwenden, in welcher mehrere digitale Objekte in einem Pool zusammengefasst werden. Somit wird die Anzahl an kostenpflichtigen Transaktionen auf der blockchain minimiert, wobei trotzdem die Effizienz gewahrt wird. Die Idee zu diesem Vorgehen stammt aus der derzeitigen Pandemie, in der die Testkapazitäten optimal genutzt werden müssen. In der Evaluation berücksichtige Ich die verursachten Kosten und die Effizienz einer Pooling-Strategie im Vergleich zur individuellen Strategie. Ich zeige, anhand des OpenPreserve Format Korpusses, dass die Anzahl an benötigten Fixitäts Tests um das 2.25-Fache und die Kosten um das 5-Fache durch eine Pooling-Strategie reduziert werden können. Ich zeige zudem eine Kontext sensitive Pooling-Strategie, welche die Volatilität einzelner Dateiformate berücksichtigt und dadurch angepasste Poolgrößen anwendet. Die Kontext sensitive Methode ist ausbalancierter mit einer Testreduzierung um das 3.28-Fache und einer Kostenreduzierung um das 3.48-Fache. Der Beitrag dieser Arbeit zu diesem Forschungsfeld ist die exakte Kostenberechnung eines dezentralen Fixitätsspeichers auf der Ethereum blockchain und zusätzlich eine Methode, um die Kosten eines solchen zu reduzieren.

Abstract

This work is about securing the data integrity of a digital archive, in which cultural assets such as images or text are stored, with the help of blockchain technology. I provide an overview of digital archives and the concepts of trust, authenticity, and integrity of data in archives and a method to strengthen these concepts. In a modern age, where falsifying and manipulating data is easier than ever, it is even more important to strengthen trust in public institutions. To ensure data integrity, cryptographic methods are usually used to create so-called fixity information. Fixity is an attribute of a digital object that can be used to validate that an object has not been tampered with for a certain time interval. While generating fixity information (e.g., MD5, SHA256) is relatively easy, storing it is more difficult when you consider that anyone who can change it can also change the underlying data. In this work, I present the Ethereum blockchain as a storage medium for the fixity information. It has already been shown that the Ethereum blockchain is suitable for persisting metadata in an unchangeable manner, but the costs for an individual strategy in which the metadata of each object is persisted individually in the blockchain are immense. To counteract the costs, I utilize a pool testing strategy in which several digital objects are pooled. This minimizes the number of chargeable transactions on the blockchain while still maintaining efficiency. The idea for this approach stems from the ongoing pandemic, in which the test capacities must be used optimally. In the evaluation, I consider the costs incurred and the efficiency of a pooling strategy compared to an individual strategy. I show, using the OpenPreserve format corpus, that the number of required fixity tests can be reduced by a factor of 2.25 and the cost by a factor of 5 by utilizing a pooling strategy. I also show a context-sensitive pooling strategy that considers the volatility of individual file formats and thus applies adjusted pool sizes. The context-sensitive method is more balanced with a test reduction of 3.28x and a cost reduction of 3.48x. The contribution of this work to this research field is the exact cost computation of a decentralized fixity storage on the Ethereum blockchain and additionally a method to reduce the costs of such.

Contents

CHAPTER 1

Introduction

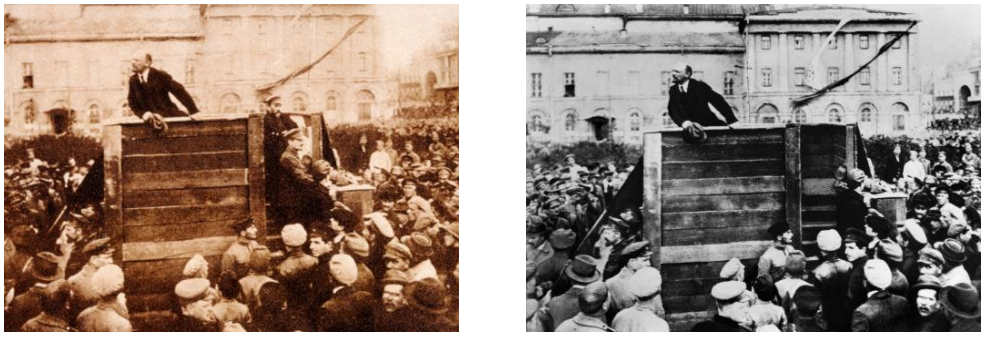
1.1 About Data Integrity

Storing cultural heritage in digital archives offers malicious actors the possibility to manipulate the data and possibly forge history. Recent digital technologies make data manipulation more efficient, less costly, and more exact and there is a long history of forging history. In 1920 a photography was taken of Vladimir Lenin atop a platform speaking to a crowd. In the original photo, see Figure 1.1a, Lenin's comrade Leon Trotsky can be seen standing beside the platform on Lenin's left side. When power struggles within the revolution forced Trotsky out of the party 7 years later, he was retouched out of the picture, see Figure 1.1b, using paint, razors and airbrushes. Soviet photo artists altered the historical record by literally removing Trotsky from the pictures [HS05, 3].

Digital archives must earn the trust of current and future digital creators by developing a robust infrastructure and long-term preservation plans to demonstrate that the archive and its staff are trustworthy stewards of the digital materials in their care [KORD10, 37]. Digital objects can be corrupted easily, with or without fraudulent intent, and even without intent at all. Data corruption is usually detected by comparing cryptographic hashes, so-called fixity information, at different time intervals [DFG+14, 1]. The object is seen as uncorrupted if the hash values are identical, since the smallest change to the object would alter the newly computed hash value immensely.

1.2 Research Problem

Although generating fixity information (e.g., MD5, SHA256) is relatively easy, managing that information over time is harder. Consider that if an actor can change the fixity information, the actor can also tamper the underlying data [KORD10, 35]. Fixity information is usually stored in databases; object metadata records or alongside content,



(a) Original (b) Retouched

Figure 1.1: Catalog Images

whereas this thesis utilizes the blockchain as a storage medium. It is shown, that the Ethereum blockchain is suited for storage of metadata, such as provenance data or fixity information

Collomosse et al. (2018) and Sigwart et. al (2020) have shown that the Ethereum blockchain, developed by Buterin et. al (2013), can indeed be utilized to ensure data integrity, but there is a problem with that: the operation cost. The cost of storing a SHA256 values on the Ethereum blockchain is about $8, which means the operation cost for an ingest of 10,000 objects costs about $80,000.

1.3 Research Goal

This thesis proposes a way to reduce the operational cost of a blockchain-based fixity information storage by applying a pool testing strategy in which several digital objects are combined in a pool to form a hash list. The idea for this approach stems from the ongoing pandemic, in which the test capacities also must be used optimally.

> Pool testing strategies build on testing a pooled sample from several patients: if the results from the pool test are negative, all patients in the pooled sample are declared not to have COVID-19; if the results of the pool are positive, each patient sample is tested individually. The pooled testing strategy is appealing, particularly when test availability is limited [CGWK20, 1].

This paradigm will be utilized in this thesis where the test specimen are cryptographic hashes, and the pool is hash list. With pooled testing, the goal is to reduce the amount of transactions, and therefore the cost by at least 50%.

1.4 Research Questions

Based on the problem described above, the following research questions arise:

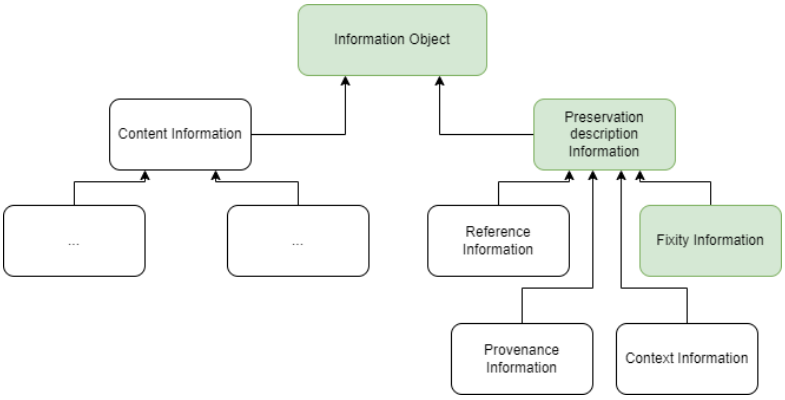


Figure 1.2: The transaction hash, used to retrieve data from the blockchain, is stored in the fixity information of the PDI of an information object [Lee10, 7]

RQ1 What is the optimal pool size based on the change rates of digital objects in the archive regarding cost and efficiency?

RQ2 To what extent can pooled testing increase the efficiency and reduce cost for a fixity information storage service on the Ethereum blockchain?

RQ3 Given that metadata has a higher change rate, what effect has the split of metadata and objects on the operation cost?

1.5 Research Approach

This work will follow the principles of Design Science Research, from [Hev07]. The artifact is designed based on a literature review of the following main topics: Diplomatics of Digital Records, Ethereum Blockchain and Pooled testing.

The application environment is a digital archive that manages any kind of data, such as images or text. Data is referred to in the archive as an Information Object (IO) and each object is provided with metadata to guarantee readability in the future, as described in the OAIS reference model. I assume that the data integrity must be ensured on ingest where a cryptographic hash value of the information object is computed and stored on the blockchain. The "location" of the fixity information on the blockchain is stored in the Preservation Description Information (PDI) in the form of a unique transaction hash, see Figure 1.2.

The *Artifact* is a combination of decentralized application on the Ethereum blockchain and a local pool structure in the archive. The smart contract exposes functions for managing cryptographic hashes of pools. The artifact must provide the following functionalities:

- Create, Read, Update SHA256 values of pools.
- Must hold a reference (ID) for the respective pool in the archive.

- Must be tamperproof for unauthorized calls.

The smart contract complements the local pool structure in the archive where fixity information of certain object pools may be requested on any given time. The artifact should reduce the amount of operations needed to ensure the integrity of 10,000 objects from ingest to retrieval process by at least 50%.

1.6 Structure of the work

The remainder of this work is structured as follows:

2) **Related Work** presents the state of the art of projects that utilize the Ethereum blockchain as a storage medium for metadata

3) **Diplomatics of Digital Records** gives an overview of the abstract concept of trust in digital archives

4) **Ethereum Blockchain** gives an overview of the concepts and functions of the Ethereum ecosystem

5) **Fixity Storage** contains an introduction to fixity information and how to handle it, this chapter also contains the implementation and deployment of the fixity storage utilized in this work

6) The **Evaluation** chapter contains the answers to the research questions

7) I interpret and explain the results in the **Discussion** chapter

8) **Conclusion** suggests future work and my contribution to the digital preservation research field

CHAPTER 2

Related Work

2.1 Project Archangel

Project ARCHANGEL is a decentralized fixity information developed in the UK with the participation of three other countries, Australia, Estonia and Norway. Their approach is to store a cryptographic hash of every incoming object of the archive on a proof-of-authority blockchain. The project started in 2017 and ended in August 2018. The goal of the project was to ensure the integrity and authenticity of digital objects in archives with the usage a private fork of the Ethereum network. Their design philosophy of operating a private blockchain has the flaw of giving a few entities the power to alter data on the blockchain. Currently, their implementation uses smart contracts as a gateway for writing to the Blockchain [CBB+18, 4].

The public's trust in Archives and Memory Institutions (AMIs) has eroded, according to the authors of the project ARCHANGEL, due to the ease with which forgery and unauthorized modifications to electronic records can be carried out due to advances in technology and the creation of numerous types of composited content. Unlike in the past, when archives relied on specific firms' products and technologies, blockchain introduces a whole new paradigm of openness and expandability. The blockchain grants permission to write records in a distributed ledger to only authorized institutions, whereas permission to view the recorded content is granted to every node participating in the blockchain. Furthermore, scalability permits the use of diverse open-source tools and the assurance of record integrity by several parties through a consensus mechanism rather than by a single centralized organization. [WY21, 4]. The proposed architecture of ARCHANGEL can be seen in Figure 2.1 where the cryptographic hash of an incoming object is computed and then persisted onto the private blockchain. After a certain time interval, the object can be retrieved from the archived and a hash will be recomputed with the same cryptographic hash function. The object is guaranteed to be unaltered if the local and online hash value are the same. I agree with their vision of publicly available fixity information, where

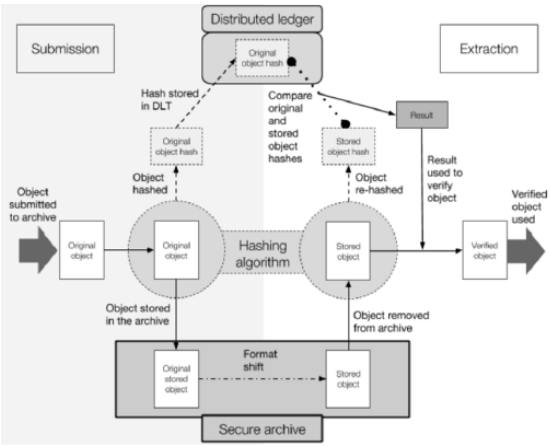


Figure 2.1: Architecture of the ARCHANGEL platform [CBB+18, 2]

everyone with an Ethereum client is able to validate the integrity of objects in archive, but there are two points which do not conform with my vision. First and foremost is the usage of a private fork of the Ethereum network that is operated by a private set of nodes, which is basically a proof-of-authority consensus mechanism and contradicts the vision of a decentralized application [CBB+18, 3]. Second, their implementation can hardly be used on the public Ethereum blockchain instead of a private fork due to the high cost of persisting data on the Ethereum main net, which is about $8 for a SHA256 word, see Section 4.8.

2.2 Provenance framework for the Internet of Things (IoT)

Sigwart et al. (2020) proposed a data provenance framework for the IoT. Their approach is to store provenance records as so-called non-fungible assets on the Ethereum blockchain. Their prototype implements the ERC721 standard, which defines an interface for non-fungible assets or tokens (NFTs) which can be transferred by any clients (e.g., sensors, wallets), leaving a trail of provenance records created by passing the data NFT from owner to owner [SBP+20, 7]. This project is important for my thesis, because it shows that the Ethereum blockchain can be utilized as a secure storage for immutable metadata which complements processes outside the blockchain. Contrary to their implementation, I decided to not utilize the ERC721 standard since the data in my case, fixity information, does not need to be transferred from owner to owner. In order to use their implementation for fixity information instead of provenance data, you would have to deploy or "mint" an NFT for each digital object in the archive. That means, massive overhead in operation cost since "minting" is basically a deployment of a smart contract, which is one of the most expensive operations on the Ethereum blockchain, see Table 4.8.

2.3 Pooled testing

Pooled Testing was first introduced by [Dor43] as a strategy to screen numerous military recruits for syphilis during World War 2. Dorfman envisioned that instead of testing each recruit's blood specimen separately, multiple specimens could be pooled together and tested at once. Positive pools' specimens would be retested individually to determine which recruits had contracted the disease, whereas negative pools' specimens would be declared negative. Dorfman wanted to save money on testing while still identifying all syphilitic-positive candidates, so he used group testing. Because the goal is to identify all positive persons among all individuals tested, this is now known as the "case identification problem." Dorfman's case identification method can be thought of as a two-stage hierarchical algorithm. Individuals from positive pools are tested in the second stage after non-overlapping pools have been screened in the first.

In this thesis, two strategies of pooled testing are implemented. First, two stage hierarchical pooling strategy, where in the first stage of this protocol N pools get initialized and filled with samples of the population. If the combined result of the pool is negative than no second stage is needed, but when a pool is declared positive a second stage is needed and all individuals from this pool have to be retested in order to find the corrupted individual. The expected number of test is equal to the number of tests in the first stage added to the number of tests in the second stage [NEG$^+$21, 3]. Second, context-sensitive pooling, where homogeneous pool samples are grouped where it is assumed that the within-group alteration rate is smaller than the between-group alteration rate. Hence, if one member of a pooled group is corrupted, there is a high likelihood that other group members are also infected [DBK20, 3].

2.4 Summary

The blockchain related projects presented in this chapter have a common vision, which is a decentralized storage for metadata to validate the integrity and authenticity of data used in applications. The ability for the public to validate that an authority did not tamper with the data in their care is important and the removal of trust strengthens the trust in public institutions. More on the subject of trust can be seen in Chapter 3 Decentralization and trust aside, the cost of the fixity storage presented in this thesis is also more than relevant as later seen in Chapter 4, where the cost and computational effort of interacting with the Ethereum network is presented.

CHAPTER 3

Diplomatics of Digital Records

3.1 About Diplomatics

Diplomatics is a science founded in France in the seventeenth century by Benedictine monk Dom Jean Mabillon in his dissertation De Re Diplomatica Libri VI (1681) to determine the provenance and authenticity of evidence attesting to patrimonial rights. It was then utilized by attorneys to settle disputes, historians to interpret documents, and editors to publish medieval deeds and charters, and it evolved into a legal, historical, and philological specialty. Classic diplomatics and modern/digital diplomatics should be distinguished since these two areas of the field do not represent a natural evolution of the latter from the former, but rather operate in parallel and focus on separate topics. Modern diplomatics encompasses all documents produced in the course of any kind of business, and is defined as "the discipline that studies the genesis, forms, and transmission" of records, as well as "their relationship with the facts represented in them and with their creator, in order to identify, evaluate, and communicate their true nature."

Both classic and modern diplomatics are concerned with determining the trustworthiness of records; however, the former does so retrospectively, examining records from several centuries ago, whereas the latter is concerned not only with establishing the trustworthiness of historical information but also with ensuring the trustworthiness of records yet to be created [KORD10, 10].

A digital record is a digital component, or group of digital components, that is saved, treated, and managed as a record, or, more specifically, a record whose content and form are encoded using discrete numeric values, such as the binary values 0 and 1, rather than a continuous spectrum of values, as defined by modern diplomatics. A digital record differs from analogue and electronic records in that it is digital. The representation of an item or physical process using continuously varying electronic signals or mechanical

patterns is referred to as analogue by InterPARES[1]. An analogue depiction of an object or physical process, in contrast to a digitally recorded representation, closely mimics the original. Any analogue or digital record conveyed by an electrical conductor and requiring the use of electronic equipment to be comprehensible by a person is defined as an electronic record by InterPARES. Using the classic archive idea, InterPARES defines a record as a document created or received as an instrument or by-product of a practical activity and set aside for action or reference [Dur09, 44].

The integrity of a record is determined not only by its appearance – which can be deceiving in the case of good forgeries – but also by the circumstances of its maintenance and preservation: until proof to the contrary, an unbroken chain of responsible and legitimate custody is considered an insurance of integrity, and integrity metadata are required to attest to that. The duty for a record's authenticity changes from the creator's trusted record keeper, who must guarantee it for the duration of the record's custody, to the trusted custodian, who must guarantee it for the duration of the record's existence [Dur09, 53]. In this thesis the Ethereum network acts as a trusted custodian which guarantees that no one is able to tamper with the fixity information of digital records.

There are no originals in the diplomatic sense in the digital environment since there are no records that are the first instance of an item. When a digital record is closed for the first time, the original is destroyed and every time it opens a copy is created. However, it can be stated that each digital record is a copy in the latest version utilized by the creator, and that any version retained by the preserver is a genuine copy of the creators record.

3.2 Trustworthiness

Whether it is a stack of paper or a digital file, trustworthiness is a concept and an obligation that lasts the life of the document. As files go through the stages of the preservation process, from initial capture and metadata extraction to longer-term strategies like migration and rights management, the needs of born-digital objects change. Born-digital objects follow a similar path from creator to intermediary, such as a dealer or other (human or technology) agent, to archival repository staff, and eventually to storage and, possibly, ingest into a digital repository. The phases of that journey make up a digital objects chain of custody, and each one has a significant impact on the trustworthiness of the born-digital elements in a given accession.
The fundamental parts of creating and maintaining trust are the provenance of both analog and digital resources, as well as documentation concerning their storage environment, what has been done to them, and by whom. In the collection and preservation of born-digital resources, the trustworthiness of an institution, a custodian, or a document is critical [KORD10, 27].

[1] InterPARES 2, Terminology Database, `http://www.interpares.org/ip2/ip2_terminology_db.cfm` (accessed on 25 April 2022)

Modern diplomatics concerns itself with four aspects of trustworthiness: reliability, authenticity, accuracy and authentication [KORD10, 10].

3.2.1 Reliability

The content of a record's trustworthiness as a statement of fact. It is evaluated on the basis of 1) the completeness of the record, that is, the presence of all formal elements required by the legal-administrative system for that specific record to be capable of achieving the purposes for which it was created; and 2) the controls exercised on the process of creating the record, among which are those exercised on the author of the record, who must be the person competent, that is, having the authority and capacity to create the record. The creator and trustworthy record keeper, that is, the person or organization who made or received the record and kept it with its other records, are solely responsible for its reliability [Dur09, 52].

Archivist' expectations reliable sources have evolved over time, from the belief that librarians and archivists would provide researchers with verifiable evidence, to more modern understandings that reject the ideal of the reliable source and regard all texts as potentially deceptive and richly ambiguous. Data-stewards should be able to offer researchers with documentation about the provenance and acquisition of the data in their care using the methods of operation and processes created by the community and professionals, such as a provenance chain or the fixity information created on ingest of the object in question [KORD10, 32].

Open source software is favorable in terms of reliability because community members can look into the code and verify that no unwanted actions will be executed. From a technical perspective, the blockchain suits as a source of reliability since it is well accepted by the public and every transaction ever made is publicly viewable and therefore fulfills the first point of reliability: completeness of the record. Also, the source code of the Ethereum network and each update (fork) gets peer reviewed and is available on GitHub[2].

3.2.2 Authenticity

Authenticity refers to a record's ability to be trusted as a record, and is defined as the fact that a record has not been tampered with or corrupted, either accidentally or with malicious intend. An authentic record keeps the same identity it had when it was created and can be presumed or demonstrated to have kept its integrity across time. The identity of a record is made up of all the features that set it apart from other records, and it is determined by the formal components on the record's face and/or its attributes, as stated in a register entry or as metadata [Dur09, 52]. With the fixity information stored on the blockchain, it can be proven that a record in the archive has not been tampered with. The simple comparison of the hash values generated in different points in time can prove the authenticity of a record.

[2] `https://github.com/ethereum/go-ethereum`

3.2.3 Accuracy

Because record correctness was subsumed under both trustworthiness and authenticity as a notion, it was never a consideration in general diplomatics. Accuracy is defined as the truthfulness, exactness, or completeness of the data (i.e., the smallest, meaningful, indivisible pieces of information) within a record. Because of the ease with which data can be damaged during transmission over space (between humans and/or systems) and time in the digital environment, accuracy must be considered and assessed as a separate quality of a record (when digital systems are upgraded or records are migrated to a new system). As a result, the responsibility for accuracy shifts over time from the creator's trusted record-keeper to the trusted custodian [Dur09, 52]. Whether it is a bit-rot or any other unintended error during the preservation process, a cryptographic hash value will be different if any bit of the object in the archive has changed and therefore the system presented in this thesis fulfills the requirements of accuracy.

3.2.4 Authentication

Authentication is defined as a statement or an element, such as a seal, a stamp, or a symbol, attached to a record after it has been completed by a competent employee. Authentication simply assures that a record is authentic at one precise point in time, when the declaration is made or the authenticating element or entity is affixed, whereas authenticity is a quality of the record that accompanies it for as long as it remains as is. A digital signature is frequently used in the digital environment to give ultimate authenticity. Because it is tied to a full record, the digital signature serves as a seal, allowing verification of the record's origin and integrity, as well as making the record unassailable and incontestable by performing a non-repudiation function. [Dur09, 53]. Authentication in this process is not intended, since I do not use any form of digital signature to prove that a record is indeed authentic on ingest. In this thesis, I assume that every object is authentic and reviewed by a competent employee.

3.3 Summary

The concept of trust in a public authority is presented in this chapter and acts as a foundation for designing the fixity storage. The concept of authenticity is implemented through the usage of SHA256 values in this thesis, where the collision resistance of the SHA256 algorithm guarantees that no object in the archive can be tampered with. Accuracy is also guaranteed through the usage of a cryptographic hash function, since the hash is completely different if a bit level error happens over the course of the preservation. Reliability is enforced by the nature of the blockchain, where every state changing action is documented immutable as long as the blockchain exists, as later presented in Chapter 4. Each of the core concepts of trust is needed in the modern digital world, where it is easier than ever to manipulate data and opinions.

CHAPTER 4

Ethereum Blockchain

4.1 Introduction to Ethereum

Open source blockchain networks like Ethereum and Bitcoin are software kits that allow you to create an economic system in software, replete with account management and a native unit of exchange that can be used to transfer funds between accounts. These native units of exchange are referred to as coins, tokens, or cryptocurrencies, but they are the same as tokens in any other system: they are a type of money that can only be used within that system to pay peers or run programs on the Ethereum network. When you want to make one of these peer-to-peer networks accessible through a web browser, you need to use special software libraries such as Python Web3[1] which is utilized in this thesis to connect to the Ethereum network [Dan17, 2].

The fact that everyone in the world can interact with the blockchain, given you have internet access and an according device, such as a PC or mobile phone, is advantageous for a fixity storage such as presented in Chapter 5. Anyone is able to validate the integrity of an object with the right resources, e.g. the transaction in which an object's fixity information was originally stored. Picture a digital archive which releases multiple objects to the public and their respective fixity information on the blockchain, the community could then validate if the object in question is authentic. Pure trust, that an archive or its stewards did not tamper with the data in their care is therefore not needed anymore.

4.2 About the Ethereum blockchain

A block is a time unit that contains a specific amount of transactions. Transaction data is captured throughout that time period, and when the unit of time expires, the next block begins. The blockchain is a representation of the history of state changes in the

[1] `https://github.com/ethereum/web3.py`

Ethereum network [Dan17, 43]. The Ethereum network divides transactions and state changes into blocks, which are then hashed. Before the following canonical block may be added on top of it, each block is inspected and validated. Nodes on the network will no longer need to evaluate the trustworthiness of every single block in the Ethereum network's history, but only validate the most recent block in the chain. Therefore, it gets harder to tamper with past transactions with each block set on top of the blockchain. The blockchain is made up of all the blocks strung together, including the genesis block, which is the first block in the blockchain. The blockchain is also known as a distributed ledger or Distributed Ledger Technology (DLT). The term ledger is correct since the chain contains every transaction in the network's history, thereby turning it into a massive, well-balanced ledger [Dan17, 55].

4.3 Ethereum Virtual Machine (EVM)

The physical manifestation of the EVM exist as a single entity sustained by thousands of connected computers running an Ethereum client[2]. In the Ethereum context, a virtual machine is a massive global computer made up of constituent nodes, which are themselves computers. In general, a virtual machine is a computer system that is emulated by another computer system. These emulations are based on the same computer architectures as the target of their emulation, but they usually reproduce that architecture on hardware other than that for which it was designed. Virtual machines can be built using hardware, software, or a combination of the two. It is both in the case of Ethereum. Rather than securing thousands of individual machines, Ethereum takes the approach of securing one massive state machine that can span the entire globe [Dan17, 48].

The EVM can run any computer program written in Solidity[3], the native programming language of the EVM. These programs will always create the same output, with the same underlying state changes, when given a specific input. As a result, Solidity programs are entirely deterministic and guaranteed to execute, as long as you pay enough for the transaction. Solidity programs are Turing complete in the sense that they can express all tasks that can be performed by computers. This means that every application executed on the platform is run by the entire dispersed network, every node [Dan17, 50].

The EVM is also a runtime environment for small programs that can be executed via a network, from the perspective of a software developer. Smart contracts are compiled into the EVM's native language, the EVM bytecode. By using a client application like Truffle[4] or Geth[5], Solidity, a high-level language, is compiled into bytecode and posted onto the Ethereum network [Dan17, 51]. I have decided to utilize Truffle for this thesis because of its fast and easy way to interact with the Ethereum network and its subnetworks.

[2] https://ethereum.org/en/developers/docs/evm/
[3] https://github.com/ethereum/solidity
[4] https://trufflesuite.com/
[5] https://geth.ethereum.org/

4.4 Smart Contracts

Smart contracts, similar to the concept of classes in traditional object-oriented programming, are the building blocks of decentralized apps operating on the EVM. When developers talk about developing smart contracts, they usually mean writing applications in the Solidity programming language for execution on the Ethereum network [Dan17, 10]. In my thesis, the fixity storage is often referred to as Decentralized Application (DAPP), which is an application built with several smart contracts. The smart contract implemented in this thesis is further described in Section 5.3.

4.5 Persistence

Decentralized storage systems, unlike centralized servers managed by a single corporation or organization, are made up of a peer-to-peer network of users that each hold a share or the whole amount of the data, resulting in a resilient file storage sharing system. All transactions in Ethereum are stored on the blockchain, a canonical history of state changes stored on every single Ethereum node [Dan17, 12]. The chain is steadily growing and on March 19, 2022, the Ethereum chain is at 602.95 GB[6], which every node on the network needs to be able to store. If the chain were to expand to large amounts of data (say 5TBs) it would not be feasible for all nodes to continue to run. Also, the cost of deploying this much data to the mainnet would be prohibitively expensive due to fees[7]. My proposed solution utilizing is pooled testing in order to mitigate the ever-increasing chain size of the Ethereum network by reducing the amount of transactions needed to operate the fixity storage.

The blockchain, as a decentralized storage system also follows some rules, constrain and enforce relationships between entities. Motives to break or alter these relationships can be found in a wide range of industries, leading to bribery and corruption and making blockchain's trustless properties even more appealing to businesses than previous generations of software and networking. Shared read/write access generates considerable complexity. Depending on where the client's machine is located and regarding its internet speed, machines all over the world may experience variable latency, resulting in certain write operations arriving out of order. This becomes even more challenging when numerous parties are expected to share the blockchain equally [Dan17, 20]. The Ethereum network solves this problem by nodes executing any code included within a block independently, this is possible due to the deterministic characteristic of the Solidity programming language. This method is not only highly parallelized, but also very redundant. Despite the significant redundancy, this is a reliable and efficient technique to balance a worldwide ledger [Dan17, 50].

Ethereum requires a persistence mechanism in order for a piece of data to last indefinitely. When executing a node, for example, the persistence method requires that the entire

[6] https://ycharts.com/indicators/ethereum_chain_full_sync_data_size
[7] https://ethereum.org/en/developers/docs/storage/

chain be considered. The chain continues to grow as new pieces of data are added to the end, requiring each node to reproduce all the embedded data. The term for this is "blockchain-based persistence." The problem with blockchain-based persistence is that the chain could grow to be far too large to maintain and retain all the data in a reasonable amount of time (e.g. many sources estimate the Internet to require over 40 Zetabytes of storage capacity).

In addition, the blockchain must have some sort of incentive system for the participating nodes in the network to validate new blocks in the blockchain. Validating a block is a computational hard task which requires nodes to have a vast computing power which is used in a consensus mechanism called proof-of-work. In this consensus mechanism a cryptographic puzzle has to be solved, by the node which aims to validate a block, in order for a block to be seen as valid. The node which solves the puzzle and therefore presents the proof-of-work is rewarded with the so-called block-reward in form of Ether tokens. Ether is the name of the token, or cryptocurrency which is mainly used in the Ethereum ecosystem to pay for transactions or to fuel smart contracts. Ether can be publicly traded into other currencies such as EUR or USD.

For blockchain-based persistence, the incentive system includes a payment made to the miner, the payment varies with the amount of data which has to be included in a block[8].

4.6 Test Networks

There are multiple independent networks implementing the Ethereum protocols, since the Ethereum source code is openly available everyone is able to set up an Ethereum network, which is often called a private fork. Networks can also be utilized as different Ethereum environments for development, testing or production use cases. The private key for an Ethereum account works across various Ethereum networks, but the transaction history or account balance is independent for each network, meaning you can not transfer funds from a test network to the mainnet and vice-versa. For testing purposes, I have utilized the Ropsten test network to deploy and interact with the fixity storage presented in Section 5. Interactions on the test network also requires you to have some funds in your wallet to pay for transactions, although the funds on the test network does not require you to gather them from exchanges for money. The Ether token for the Ropsten test network can be acquired through public faucets, where you receive some tokens after you have input your wallet address. Some faucets require you to have a Twitter account (e.g. `https://faucet.paradigm.xyz/`), with a certain amount of tweets and follower for bot protection and some faucets do have a large waiting queue for Ether tokens, in the case of `https://faucet.dimensions.network/` the waiting time can reach up to three days. Faucet are usually time gated, meaning that you could request Ether tokens only once per day. At small scale, this limitation is no problem for this thesis because persisting around ten pools on the blockchains require at max 0.002 Ether tokens. The one per day paradigm gets problematic for the experiment including 10,000

[8] `https://ethereum.org/en/developers/docs/storage/`

objects, which costs about 20 Ether tokens, which means I would have to request tokens for 20 days straight to perform one experiment on a real live environment such as the Ropsten test network. I have utilized each before presented faucet to gather funds for the experiment, my suggestion for future work is to use the following Ropsten test network faucet: `https://faucet.egorfine.com/`. This faucet does require you to either have an active social media account, neither does it implement a waiting queue. There are multiple options to choose from for a testnet, from the most prominent networks presented at the official Ethereum documentation[9] only the Ropsten test network implements the proof-of-work algorithm, which means it is the best representation of Ethereum to date, and for that reason I decided to utilize the Ropsten test network in my thesis.

4.7 Gas and Fees

Gas refers to the unit that measures the amount of computational effort required to execute specific operations on the Ethereum network, this property allows you to calculate the operation cost for the fixity storage without taking the price fluctuation of ETH into account. Since each Ethereum transaction requires computational resources to execute, each transaction requires a fee. Gas fees are paid in Ether tokens. Gas prices are denoted in Gwei, which itself is a denomination of ETH - each GWEI is equal to 0.000000001 ETH. For example, instead of saying that your gas costs 0.000000001 Ether, you can say your gas costs 1 Gwei. The word "Gwei" itself means "giga-wei", and it is equal to 1,000,000,000 wei. Wei itself is the smallest unit of the Ether token[10].

Miners are paid in Ether tokens for validating blocks, and also for running scripts on the network. The cost associated with electricity consumption of nodes running the Ethereum network is one of the factors that gives Ether, as a cryptocommodity, its intrinsic value—that is, someone paid real money to their electricity company to run their mining machine. Specialized mining rigs, which use arrays of graphics cards to increase their odds of solving the cryptographic puzzle in the proof-of-work mechanism, and therefore completing a block and collect the block-reward [Dan17, 12]. Mining creates the consensus needed to make legitimate state changes in the distributed ledger, and miners are compensated for their contributions. This is how Ether tokens and Bitcoins are created [Dan17, 57]. To ensure that the system is not clogged by meaningless spam contracts, there must be a fee associated with each instruction the EVM executes. An internal counter maintains the amount of the fees incurred each time an instruction is executed, which are then charged to the user. When a user initiates a transaction, a tiny percentage of the user's fund is set aside to pay these fees [Dan17, 58].

The costs are the driving factor of my thesis where I try to make as few transactions as possible to run a fixity storage on the Ethereum network. The fees are determined by the transaction's gas cost; gas is a unit of effort, not a subcurrency, therefore it cannot be held or hoarded. In computer words, it simply assesses how much work each step

[9] `https://ethereum.org/en/developers/docs/networks/`
[10] `https://ethdocs.org/en/latest/ether.html`

Figure 4.1: Illustration of an Ethereum transaction https://ethereum.org/en/developers/docs/transactions/

of a transaction will take. You need enough Ether tokens in your account to be able to pay for the transaction, or else it will get reverted. There is no need to buy gas individually because there is no gas token. Gas prices ensure that network computing time is adequately priced and varies with the time of the day, meaning the more traffic there is on the network the higher are the gas prices for transactions or smart contract calls [Dan17, 59]. The only person who suffers if you transmit a set of computationally difficult commands to the EVM is you, it has no bearing on the transactions of others. There is no method to jam the Ethereum Virtual Machine without spending a significant amount of money in transaction fees. The gas price system serves as a de facto means of scalability. Miners have complete control over which transactions they will include in a block, depending on which transaction pays the highest fee rates, as well as the block gas limit. The block gas limit sets the maximum amount of processing and storage that can be done per block [Dan17, 60].

4.8 Transactions

An Ethereum transaction refers to an action initiated by an externally-owned account, in other words an account managed by a human, not a contract. For example, if Bob sends Alice 1 EVM, Bob's account must be debited and Alice's must be credited. This state-changing action takes place within a transaction. It is a mechanism for an external account to provide the EVM instructions to do a task. To put it another way, it is a method for an external account to send a message to the system. A transaction in the EVM is a cryptographically signed data package storing a message, which tells the EVM to transfer tokens, create a new smart contract, interact with an existing one, or perform some calculation. Contract addresses, like users with external accounts, can be transaction recipients [Dan17, 60]. Transactions that affect the EVMs state must be broadcast to the entire network. A miner will execute the transaction and propagate the resulting state change to the rest of the network once any node broadcasts a request for a transaction to be conducted on the EVM[11]. Transactions, which do not alter the

[11] https://ethereum.org/en/developers/docs/transactions/

global state are free of charged and are referred to as Calls. As seen in the source code in Listing 5.1, *getPoolHash(uint32 poolId)* is marked with the keyword **view**, which means the function itself does not alter the state of the network. Contrary to the function *setPoolHash(uin32 poolId, bytes32 poolHash)* which does alter the networks state and therefore generates cost for the fixity storage. In this thesis, writing actions on the blockchain are often referred to as Transactions; and reading actions are referred to as Calls. Table 4.8 presents the gas cost of the most relevant network operations used in this thesis according to the Ethereum yellow paper [Woo14, 27].

Operation Name	Gas Cost	Description
codedeposit	200	gas cost per byte the compiled smart contract
txcreate	32000	create a new smart contract
transaction	21000	base cost for a transaction
txdatazero	4	gas cost for every zero byte of the transaction data
txdatanonzero	16	gas cost for every non-zero byte of the transaction data
sset	20000	set a persistent variable for the first time in the contract
sreset	2900	the cost for updating a persistent variable

4.9 Summary

The learnings in this chapter is on how to interact with the Ethereum network, which tools to use and how the computational cost comes about. I have presented that it is important to prune all unnecessary transaction and storage data to mitigate the ever-increasing chain size of the Ethereum blockchain and reduce the cost for the user of the fixity storage presented in Chapter 5.

CHAPTER 5

Fixity Storage

5.1 Fixity Information

Fixity Information provides the data integrity checks or validation/verification keys used to ensure that the particular Content Information object has not been altered in an undocumented manner[Lee10, 8]. In this thesis, fixity information will be a transaction hash, with which the fixity information can be retrieved from the Ethereum blockchain. With fixity information you are able to answer questions regarding the authenticity and integrity of an object, which supports assertions about the authenticity and trustworthiness of digital objects [KAN+17, 3].
Have you received the files you expected? When fixity information is provided with objects upfront, it can be used to validate that you have received what was intended for the collection.
Is the data corrupted or altered from what you expected? Once you have generated baseline fixity information for files or objects, comparing that information with future fixity check information will tell you if a file has changed or been corrupted.
Can you prove you have the data/files you expected, and they are not corrupt or altered? By providing fixity information alongside content, you enable your users to verify that what they have is identical to what you say it should be.

Fixity is a key concept for the long-term preservation and management of digital material for many reasons. Previous scholarship on fixity has shown its vital importance in discovering changes to data and all that this error-checking can imply: authenticity and renderability of files, trustworthiness of institutions, and system monitoring/maintenance. Despite the centrality of fixity to the field of digital preservation, there is little prescriptive guidance on when and how to create fixity information, where to store it, and how often to check it. This absence is not without reason, however: the incredible variety of organizational structures, priorities, staffing levels, funding, resources, and size of

collections held by institutions that do digital preservation make it difficult to establish a single set of one-size-fits-all best practices [KAN+17, 38].

5.1.1 Generating Fixity Information on Ingest

It is important to check the fixity of content transferred to you when you bring it under your stewardship. Whenever possible, it is ideal to encourage content providers or producers to submit fixity information along with content objects. You can only provide assurance about the fixity of content overtime once you have initial fixity values, thus it is imperative to document fixity information as soon as possible. If fixity information is not provided as part of the transfer, you should create fixity information once you have received the materials [KGA+14, 4].

5.1.2 Fixity Checks

Most fixity procedures involve a computational method that takes a digital file as input and outputs an alphanumeric value; this output value is used as a baseline comparison each time the fixity check is rerun [KAN+17, 5] In addition to checking fixity before and after transfer, collections of digital files and objects should be checked on a regular basis. There are a range of systems and approaches focused on checking the fixity values of all objects at regular intervals. This could be monthly, quarterly, or yearly for example. The more often you check, the more likely you are to detect and repair errors [KGA+14, 4]. The information must be put to use, in the form of scheduled audits of the objects against the fixity information. Additionally, replacement or repair processes must be in place. Ideally these will have been tested before being needed. All of this is critical for bit-level preservation, but ensuring fixity does not mean that the object is or will be understandable. Long term access is also contingent on ones ability to make sense of and use the contents of the file in the future [KGA+14, 2]. The rate of fixity checking is going to be dependent on how quickly you can run the checks, the complexity of your chosen fixity instrument, and how much of your resources (e.g., CPU, memory, bandwidth) can be used for this operation. This can become a choke point as the amount of digital content increases but the infrastructure to perform the checks stays the same. In a situation like this, the fixity checking activities can adversely affect other important functions like delivery of the content to users.

5.1.3 Fixity Instruments

For a given fixity instrument, see Table 5.1, the harder it is to find two objects that result in the same fixity information, the more "collision resistant" that instrument is. This is important mostly for preventing the concealment of intentional changes to objects. For example, expected file size and expected file count are extremely vulnerable to collision: it is very easy for someone to replace an object with one that matches in file size. It is also possible (although unlikely) for an unintentional change (such as corruption or human error) to result in an object with the same fixity information for instruments that

Fixity Instrument	Definition	Level of Effort and Return of Investment
Expected File Size	File size that differs from the expected can be an indicator of problems, for example by highlighting zero byte files	Low level of effort and low level detail. File size is auto-generated technical metadata that can be viewed in Windows Explorer or other common tools.
Expected File Count	File count that differs from the expected can be an indicator that files are either added or dropped from the package.	Low level of effort and low level detail. File count is auto-generated technical metadata that can be viewed in Windows Explorer or other common tools.
CRC	Error detection code, typically used during network transfers.	Low level of effort and moderate level of detail. CRC function values, which are variable but typically 32 or 64 bit, are relatively easy to implement and analyze.
MD5	Cryptographic hash function	Moderate level of effort and high level of detail. CPU and processing requirements to compute the hash values are low to moderate depending on the size of the file. The output size of this hash value is the lowest of the cryptographic hash values at 128 bits.
SHA1	Cryptographic hash function	Moderate level of effort, high level of detail, and added security assurance. Due to its higher 160-bit output hash value, SHA-1 requires more relative time to compute for a given number of processing cycles CPU and processing time than MD5.
SHA256	More secure cryptographic hash function	High level of effort, very high level of detail, and added security assurance. With an output hash value of 256 bits, SHA-256 requires more relative time to compute for a given number of processing cycles CPU and processing time than SHA-1.

Table 5.1: Various Fixity Instruments [KAN+17, 6]

have low collision resistance. Of the fixity instruments described above, the cryptographic hash functions (MD-5, SHA-1, and SHA-256) are the most collision resistant; SHA-256 is recommended for applications where security is important. However, performing fixity checking and replacing damaged objects is critical for any preservation system, and using any fixity instrument is much better than none at all. Note that as the level of security of the hash function increases, so do the time and resources needed to compute [KGA+14, 7].

5.1.4 Storage Medium

There are various storages to store fixity information, each of them has some advantages and disadvantages in terms of convenience, security and throughput.
In object metadata records: In many cases, you will want to record some file or object fixity information wherever you store and manage the metadata records. These metadata records are actually stored as discrete files or in databases. This is particularly useful for maintaining originally submitted or generated fixity information as part of the long-term object metadata.
In databases and logs: For checks you run at given intervals you may not want to be constantly adding to your object metadata records. In this case, it makes sense to keep fixity information apart from the object in databases and logs that you can return to when needed.
Alongside content: Its often ideal to have fixity information right alongside the content itself. That way, if you have problems with other layers in your system, or want to transfer some set of objects, you still have a record of previous fixity values alongside your content. For example, the BagIt specification includes a requirement for a hash value for the bagged content alongside the content. Similarly, some workflows involve creating *.md5 files, which are simply text files with the md5 hash, named identically to the file it refers to, but with an additional .md5 extension.
In the files themselves: When a checksum is for a portion of a file, it may make sense to store the information directly in the file. Note that this only makes sense when storing sub-file fixity information within a file. Adding fixity information for an entire file to the file itself changes the file and therefore changes its fixity value [KGA+14, 7].
Ethereum Blockchain: In this thesis I have chosen the blockchain to be the storage-medium for the fixity information. Its immutable state and availability in addition to its novel use-case in digital archives have influenced my decision.

5.2 Example of a fixity storage

A fixity storage must persist any kind of fixity information, in my thesis SHA256 values, for long term and guarantee that the persisted content is unaltered until retrieval of the content. I have chosen the Ethereum network as a medium for persisting file fixity information, see Section 5.1.4 for my reasoning. In Figure 5.1 the lifecycle of fixity information is presented, where at some point in time the information is ingested into

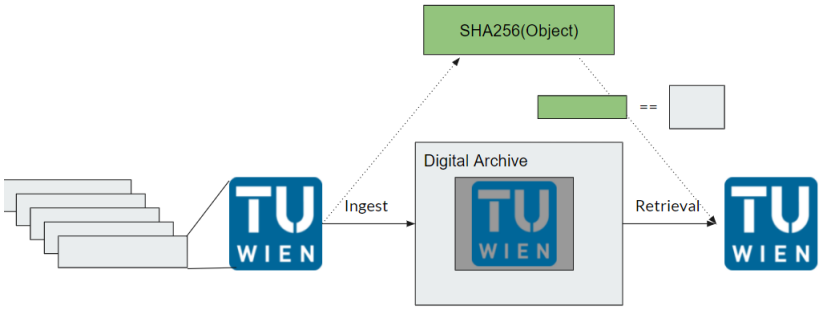


Figure 5.1: Example of a fixity storage

the storage and after a certain time interval the information is fetched and compared to the retrieved SHA256 value from the digital object of the archive. If both cryptographic hashes match, the object is guaranteed to be unaltered.

5.3 Implementation

The fixity storage presented in this thesis is implemented in Solidity, a programming language designed for the EVM, see Section 4.3. The basic functionality of the smart contract, as described in Section 1.5 can be seen in the source code presented in Listing 5.1

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity >=0.4.22 <0.9.0;
3
4  contract FixityStorage {
5    mapping(uint32=>bytes32) pools;
6    address creator;
7
8    constructor()public{
9      creator = msg.sender;
10   }
11
12   function getPoolHash(uint32 poolId) public view returns(bytes32) {
13       return pools[poolId];
14   }
15
16   function setPoolHash(uint32 poolId, bytes32 poolHash) public {
17     require(msg.sender==creator);
18     pools[poolId]=poolHash;
19   }
```

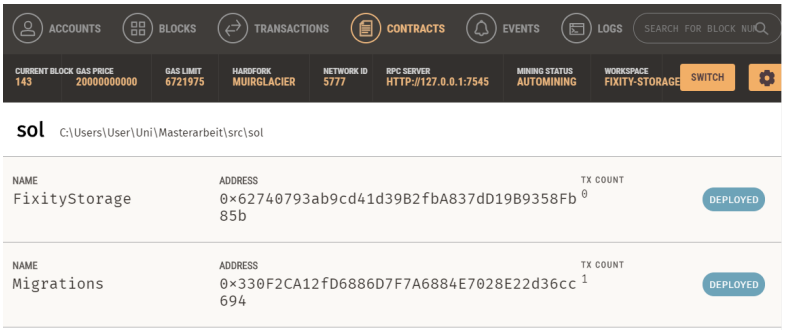


Figure 5.2: Ganache, an interactive user interface for the Ethereum blockchain.

```
20 }
```

Listing 5.1: MVP source code of the fixity storage deployed on the Ropsten test network `https://Ropsten.etherscan.io/address/0x0243c7aa552730E8C6F7ED25A480a7C0c88a70f0`

where *getPoolHash(uint32 poolId)* implements a read function; *setPoolHash(uint32 poolId,bytes32 poolHash)* implements create and update function. The mapping type is native in Solidity which implements a hash map consisting of a key and a value, where in this case the key is an integer representing the *poolId* and the value is a *bytes32* object representing the SHA256 root hash of the pool. The *poolId* is the reference to the local pool in the digital archive, with which fixity information can be retrieved for a certain pool from the contract. The Solidity language presents a convenient method to prevent unauthorized calls to the setPoolHash() method, which is *require(msg.sender==creator)*. The native method *require* is a "guard" function which improves the readability of the smart contract code and reverts the instruction if the condition is not met. The condition in this case is, that only the creator, which is set in the constructor, of the fixity storage is able to create and alter the information stored on the blockchain.

5.4 Deployment

I utilized truffle[1] to run my deployment of the smart contract, it brings built-in smart contract compilation, linking, deployment and binary management with automated contract testing. The reasoning behind my decision is that, truffle has all the tools needed to implement a smart contract in one package and therefore reduced complexity in the development process. In the first iteration of the development process, I used the graphical user-interface Ganache[2] to get a better feeling for the Ethereum blockchain, see Figure 5.2. It allows you to navigate through your smart contract in a graphical user interface (GUI) and look at the state variables or functions to validate that your smart contract was

[1] `https://trufflesuite.com/index.html`
[2] `https://trufflesuite.com/ganache/index.html`

successfully deployed. Ganache has a massive disadvantage when doing high throughput computing: it seems that it is not suited to withstand 10000 transactions built in a python for loop. Therefore, I only use the GUI it in the beginning of the experiment where I only persisted about 10 objects at a time without problems. For high throughput computing, truffle offers a command line tool to interact with the blockchain. The command line tool is resistant and shows no weakness when persisting 10,000 objects. Truffle also allows to config other networks, e.g. the Ropsten test network, which can be defined as a parameter in the deployment process. The deployment requires to have some Ether tokens in your account to pay the miner to integrate your smart contract in a block. Truffle offers some utility regarding automated deployment, through the usage of deployment configurations where you can set the gas limit and the network configurations. Truffle offers a migrations feature, which is responsible for staging and versioning deployments. The migrations feature requires to have a smart contract deployed on the blockchain, which causes additional cost (191943 gas) to the fixity storage, therefore I have decided not to use the migrations feature, as it is controversial in the community and seen as unnecessary traffic and cost[3]. The deployment cost of the smart contract can be calculated as follows:

$$\begin{aligned} C_{deployment} &= transaction + txcreate + codedeposit + txdatanonzero + txdatazero \\ &= 21,000 + 32,000 + 200 \cdot 832 + 226 \cdot 4 + 800 \cdot 16 \\ &= 233,104 \end{aligned} \tag{5.1}$$

where *transaction* is the base cost for a transaction; *txcreate* is the operation used to create a smart contract; *codedeposit* is the gas cost for each byte of the runtime bytecode of the smart contract, which is 832 bytes and can be seen in the Input Data field of the transaction on `https://ropsten.etherscan.io/` with the following hash: *0x844f76cfff6e00f29487f6fe3c99d8a69eab576e7c190e5b392745b48924a1f6*. $G_{txdatanonzero}$ is 16 gas for each non-zero byte in the compiled bytecode of a transaction;$G_{txdatazero}$ is 4 gas for each non-zero byte in the compiled bytecode of a transaction; and $Contract_{bytesize}$ is the size of the compiled bytecode of the contract, see Section 4.7 for the exact gas amount for each transaction. The amount of gas consumed by the deployment of the decentralized fixity storage is 166,079 gas on the Ropsten testnet. The transaction used to deploy the contract can be found on `https://ropsten.etherscan.io/` with the following hash: *0xf383c4bf0a5c32dd3369b02f68fd4e4400ef59343ad472bc96a28827f32c9abb*, where additional information, such as deployment date or the blocknumber, can be read. The address of the fixity storage, presented in Listing 5.1 is *0x0243c7aa552730E8C6F7ED25A480a7C0c88a70f0*.

5.5 Authorized Access

In Solidity one can require a certain address to access a function, the keyword *requires* can be used so that a transaction from an unknown address can be reverted and only the owner of the creator address can update the mapped SHA256 values in the smart contract.

[3] `https://github.com/trufflesuite/truffle/issues/503`

An interesting topic is also how the private key in the archive may be managed, which is not part of this thesis, my suggestion for future work is a multisignature wallet which can be used to split the responsibility of the contract owning Ethereum account, the creator account. Since the entity which controls the private key of the creator account is able to create and update fixity information on the blockchain. In the case of a compromised creator account, each entry of fixity information may be invalidated by the archive and new fixity information have to be uploaded to the blockchain. The key user management of the creators address may be investigated in further research. For this thesis I assume that the private key of the creator account is well managed and each transaction coming from the creator's address is legit.

5.6 Cost of interacting with the fixity storage

The cost of interacting with the fixity storage depends on the desired action. There are three different ways to interact with the contract: (1) create a new entry (2) update an existing entry and (3) read an entry. The cost of a transaction, which invokes the *setPoolHash* function and ultimately stores 256 bit on the blockchain can be calculated with Equation 5.2

$$\begin{aligned} C_{setPoolHash} &= gasAmount \cdot gasPrice \cdot ethPrice \\ &= 42,368 \cdot 0.00000005 \cdot 4,000 \end{aligned} \tag{5.2}$$

which results in $8.47 on average for an ETH price of $4000. The *gasAmount* in Equation 5.2 can be calculated as follows:

$$\begin{aligned} gasAmount &= transaction + sset + txdatazero + txdatanonzero \\ &= 21,000 + 20,000 + 16 \cdot 68 + 4 \cdot 70 \\ &= 42,368 \end{aligned} \tag{5.3}$$

where *txdatazero* is the number of zeros in the transaction data and *txdatanonzero* is the amount of non-zero bytes in the transaction data. The transaction data for Equation 5.3 can be found in the *Input Data* field of the transaction on Etherscan[4]. This field contains the binary data that formed the input to the transaction, the structure is depending on which type of transaction is executed. Whether the transaction is contract call, contract deployment or message call the input data holds instructions in form of bytecodes to properly execute the desired interaction. The hexadecimal form of the input data is used for the calculation of *txdatazero* and *txdatanonzero*.

The cost for updating an existing entry can be calculated in the same way as for creating ones, with the difference that the costly *sset* operation can be spared and therefore updates costs 20,000 gas less than creates. Reading an entry is free of charge, since no state changes has to be forwarded to the blockchain. To explain the cost of the non-zero

[4] `https://ropsten.etherscan.io/tx/0x8839e03f0143bad34060fa909c35d30f2edb22dd4fdac0264de8a`

and zero bytes can be explained by looking the following *Input Data* as a hexadecimal value:

```
0x178d2929000000000000000000000000000000000000000000000000000000000
000000000001760d27083f6e2d1c46a65938c03a0c52dccf55cb4eb68a720e6
efe3a8851f78
```

where zero and nonzero bytes are counted and multiplied by their respective gas cost, which in this example is $txdatazero + txdatanonzero = 4 \cdot 70 + 16 \cdot 68 = 1368$. The amount of non-zero bytes can be interpreted as the amount of effort given to the network.

5.7 Proof of Concept

The proof of concept is written in Python in form of a Jupyter Notebook. It can be found on GitHub[5]. With Python's Web3.py I get the contract from the blockchain and extract its functions *setPoolHash* and *getPoolHash*. For the live experiment on the Ropsten test network I have initialized 100 digital objects, in form of SHA256 values, with a change rate of 0.1, meaning that %10 of the ingested objects will be changed during the storage in the archive. The low amount of objects in the live experiment is due to the fact, that the Ropsten Ether token is hard to get, since the most relevant publicly available faucets are drained out and the ones available have a rate of only 0.1 Ether per day, as I explained in Section 4.6. I compute the optimal pool size k for N=100 and p=0.1 with Equation 6.6 and created $\lceil N/k \rceil$ pools. Each pool is assigned a unique identifier and holds an array of objects, from which a hash list is formed, and the resulting root hash is later persisted on the blockchain. Before persisting the root hashes, I estimate the gas cost for a single transaction in order to set the gas limit, a transaction will revert if the cost exceeds the manually set gas limit. I utilized Equation 5.3 and added 20% in order guarantee that the transaction is not underpaid, which may happen if the network is congested at the time of firing the transaction. At the time of the experiment, the gas price was 86.11 gwei which is 0,0000000896ETH, resulting in $15.18 for a *setPoolHash* transaction calculated with Equation 5.2 with an ETH price of $4000. Admitted, this gas price is really high, usually the gas price is about 40 gwei. For each pool, 25 in total, I uploaded each root hash on to the blockchain and waited for the transaction to finish and run some tests to see if the transactions were successful. After uploading the root hashes, I artificially corrupted the objects in the archive with a Bernoulli trial where an object gets corrupted if a random number between 0 and 1 is below p=0.1. At last, I repaired the archive by checking the local pools with the ones on the blockchain, and if the pool hashes did match the pool is seen as uncorrupted. Where the pools with non-matching hashes got replaced by copies of the objects. To sum it up, the cost for the live experiment was for 25 writing transactions 0.11204969029 ETH ($448) and 8 data-scrubbing operations resulting in 33 total operations. Data-scrubbing is an operation, where a corrupted object in the archive gets replaced with a fresh copy. The copy is assumed to be uncorrupted with the original hash value, therefore no further writing transaction is needed when you replace

[5] https://github.com/metsch/masterthesis/blob/main/src/py/poc_ropsten.ipynb

an object in the archive. Whereas, with an individual testing strategy the amount of writing transactions would have been 100 and estimated 10 data-scrubbing operations.

5.8 Summary

In this chapter I have presented the various forms of fixity information with their respective advantages and disadvantages together with my reasoning on why I have decided to utilize SHA256 cryptographic hashes. I have also presented various storage-media for fixity information and why it is so important that these storages guarantee for immutability in regard to history forgery. The interactions with the fixity storage are tested in an experimental environment, presented in the preceding section 5.7.

CHAPTER 6

Evaluation

The proposed fixity storage is evaluated on a local installation of the Ethereum blockchain and on the online Ropsten test network, where ETH tokens used for transactions are free of charge. The dataset used for evaluation is the format-corpus[1] from Open Preserve Foundation[2] on commit *4e4b9a34540f72612ba6eab2d28bccceb7a848ae*[3] on the 16th of February.

I analyzed it to get a better understanding of various file formats, mostly on how volatile they are. The format-corpus is well-structured in a public GitHub repo with a decent amount of reputation in form of GitHub stars, where other datasets did provide corrupt links;were not available or did expect a lot of work to download them because they only allow single file downloads instead of bulks. Convenience; diversity of file extensions; and reputation of the organization affected my decision to use the format-corpus for analysis. For the analysis I used the python package folderstats[4] which transforms a directory into a pandas[5] dataframe. The repository contains 1560 files with 90 distinct file extensions, e.g. 986 *.xml* files which are mostly PRONOM[6] registry files or *pom.xml* files in case of java projects.

In Figure 6.1 you can see the distribution of file extensions where *.xml* files make 62.2 percent of the portion and PDF with 6.79 percent as the second-largest portion. To get a baseline estimate on how volatile certain extensions are due to updates and changes, I analyzed Git logs and count how many times a file extension was involved in a commit. The resulting column *positives* was always higher or equal than the occurrence of a file

[1] https://github.com/openpreserve/format-corpus

[2] http://openpreservation.org/

[3] https://github.com/openpreserve/format-corpus/commit/4e4b9a34540f72612ba6eab2d28bccceb7a848ae

[4] https://pypi.org/project/folderstats/

[5] https://pandas.pydata.org/

[6] https://www.nationalarchives.gov.uk/PRONOM/

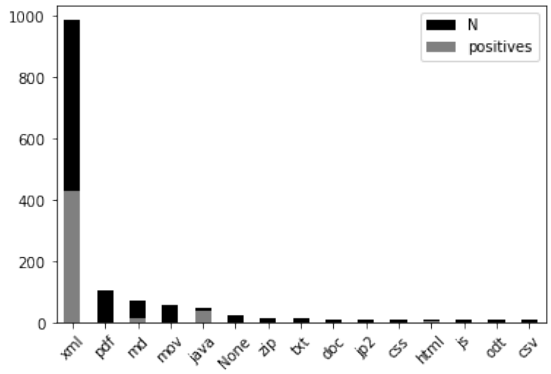


Figure 6.1: Distribution of file extensions in the format-corpus of Open Preserve Foundation. `https://github.com/openpreserve/format-corpus`

extension, since each file was involved in at least one commit, the initial commit of the file. To count the commits I used the python package gitphyton[7] and utilized the git command –log. For each file in the repository, I fired up the command *git log –oneline filename* in python which resulted in one or multiple lines of logs. Later I used the multiline log as an input for the *.splitlines()* function which results in an array of log lines, the length of this array minus 1 (the initial commit) is used to determine the new column *positives* in the dataframe which shows if and how often a certain file has changed over the lifetime of the git repository. This method of determining the volatility of a certain file extension is by no means ideal, but there is no other method to check how often a certain file has changed without monitoring them on a system for a certain time interval. Therefore, I have decided to use the amount of file changes in the git repository as a rough estimate. The change rate p of a file extension is calculated with $positives/N$.

The two key parameters for evaluation are the operation cost and efficiency. *Operation cost* T(S) is calculated by multiplying the number of relevant cost transactions with the average gas cost for a *setPoolHash* transaction on the blockchain, see Equation 6.1

$$T(S) = W(S) \cdot gasAmount \tag{6.1}$$

where W(S) is the number of write transactions on the blockchain; *gasAmount* the amount of gas used for a writing operation, see Equation 5.3. I intend to present the operation cost in the form of gas, the unit that measures the amount of computation effort required to execute specific operations, instead of the cost in USD or EUR. This is because, the amount of gas used during the experiment stays constant, whereas the price of the ETH token fluctuates heavily. Therefore, the amount of gas is a better indicator on how costly the fixity storage is. *Efficiency* E(S) is measured by the number operations needed utilizing pooled testing compared O(S_h) with the individual testing strategy O(S_i), where the efficiency of pooling strategy S is expressed by Equation 6.2

$$E(S) = O(S_i)/O(S) \tag{6.2}$$

[7] `https://gitpython.readthedocs.io/en/stable/`

where, assuming the preservation process of 10,000 digital objects without pooling requires $O(S_h)$ operations and the preservation of the same objects using strategy S requires O(S) operations. In the case of individual testing the efficiency is 1, whereas if strategy S requires two times fewer operations the efficiency E(S) is 2 [ŽLG21, 4].

What is the optimal pool size based on the change rates of digital objects in the archive regarding cost and efficiency?

Change rate p represents the change rate, e.g., when I assume that 2,000 of 10000 objects will be changed or corrupted during the preservation process, $p = 2,000/10,000 = 0.2$.

Operation Cost: The optimal pool size, which will result in the lowest number of cost relevant transactions is the number of pools J, as seen in Equation 6.3, because I only must write onto the blockchain during the ingest process where the root hashes of pools are persisted.

$$W(S) = J = \lceil N/k \rceil \tag{6.3}$$

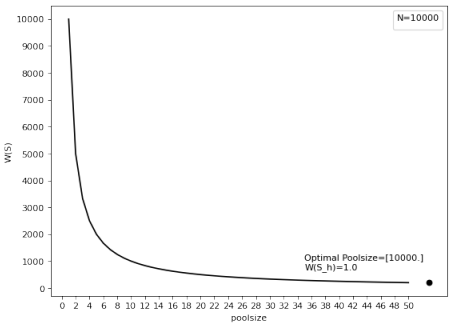
After the first iteration of the experiment, the number of corrupt objects per positive pools were included in the first draft of Equation 6.1, where I assumed that I had to re-calculate corrupt pool hashes and re-store them on the blockchain, but these "repairing" actions can be done locally through data scrubbing. I only need to know that the pool is corrupt to substitute each object in the pool with a correct copy in the archive. For this part of the research question, local operations are out of scope because they do not cause direct cost on the blockchain, therefore the optimal pool size calculated by Equation 6.1 is N, see Figure 6.2a. A pool size of N results in exactly 1 cost relevant transaction since I have combined every object on ingest into a hash-list which's root will be stored on the blockchain. This solution does not scale well, e.g., picture the process of retrieving a single object from the archive. In order to guarantee that the object is unaltered, I would have to re-compute the hash list from every object in the archive (or the bulk ingest in question). Additionally, if the single pool is corrupted, I must replace the whole bulk with copies. So, there must be an answer, which rewards smaller pool sizes to avoid too much data scrubbing. The number of data scrubbing operations is the number of objects in positive pools on retrieval and is calculated with Equation 6.4,

$$R(S) = J_+ \cdot k = ((1 - (1 - p)^k) \cdot \lceil N/k \rceil) \cdot k \tag{6.4}$$

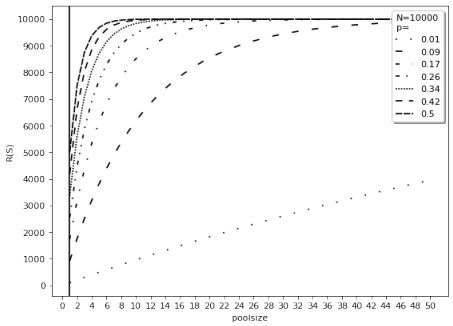
where $(1 - p)^k$ is the probability of a pool of size k being unchanged at a change rate of p and the probability of a pool being changed is $1 - (1 - p)^k$. To find the optimal pool size regarding the amount of data scrubbing operations, I had to minimize Equation 6.4, as seen in Figure 6.2b. To find the optimal pool size in terms of operation cost and efficiency, operation cost and data scrubbing has to be accounted, which results in the final Equation 6.5

$$O(S) = J + J_+ \cdot k = \lceil N/k \rceil + ((1 - (1 - p)^k) \cdot \lceil N/k \rceil) \cdot k \tag{6.5}$$

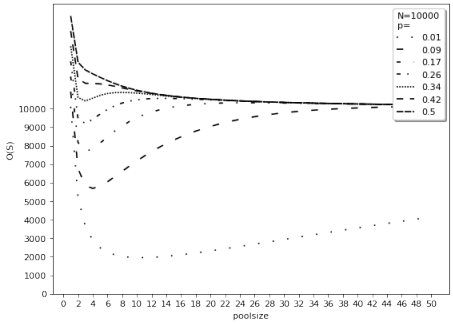
where J is the number of pools and $J_+ \cdot k$ is the number of objects in corrupted pools. Therefore, the number of expected operations is the number of writing operations on



(a) Optimal pool size regarding the writing transactions W(S).



(b) Optimal pool size regarding the data scrubbing operations R(S).



(c) Optimal pool size regarding the total operations O(S) = R(S) + W(S).

Figure 6.2: Comparison of optimal pool sizes

ingest plus the number of data scrubbing operations on retrieval. By adding the amount of data scrubbing operations, the optimal pool size gets significantly lower, see Figure 6.2c. The optimal pool size k can be determined by finding the global minimum of Equation 6.5, which results in the highest efficiency in Equation 6.2. In the experiment, I have utilized Equation 6.6 presented by [REHHR20, 3], because it is more efficient than looking for the global minimum by force, and round the pool size up in order to avoid non integer pool sizes.

$$k = \lceil 1.24 \cdot p^{-}0.466 \rceil \tag{6.6}$$

In Table 6.1 and Figure 6.3 it is shown that for ingest bulks with higher change rates smaller pool sizes are favorable[8]. The highest efficiency E(S) can be achieved when the change rate is the lowest, where larger pool sizes are favorable, see Figure 6.3.

To what extent can pooled testing increase the efficiency and reduce cost for a fixity information storage service on the Ethereum blockchain?

A context-sensitive approach in the medical field was proposed in 2020 were members of homogeneous groups were pooled, e.g. families, office colleagues or neighbors, and is

[8] https://github.com/metsch/masterthesis/blob/main/src/py/rq1.ipynb

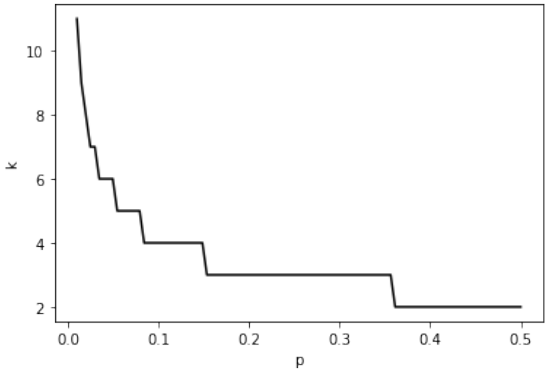


Figure 6.3: Optimal pool sizes k with change rate p

N	p	k	E(S_h)	C(S_h)
10000	0.010	11	5.16	11
10000	0.05	5	2.39	5
10000	0.09	4	1.87	4
10000	0.14	4	1.62	4
10000	0.18	3	1.50	4
10000	0.22	3	1.40	3
10000	0.27	3	1.34	3
10000	0.31	3	1.30	3
10000	0.35	3	1.27	3
10000	0.400	2	1.22	2

Table 6.1: Efficiency E(S) and Cost Efficiency C(S) of two-stage-hierarchical pooling compared to individual testing.

proven to be more effective than individual testing [DBK20, 4]. In this thesis, the idea of grouping similar files to increase efficiency of the pooling strategy, estimating their change rate and calculating the in-group efficiency as presented in Equation 6.2. In Figure 6.4, it is shown that the volatility in the repository only affects a few file extensions, where 77 out of 90 file extension where never altered and have therefore a p value of 0.00. These unaltered file extensions are grouped with a group size of $Group_N$, which results in large bulks in the ingest process that have no impact on the retrieval since I assume that no object in this particular group will be altered during the preservation process. For instance, in Table 6.2, the file extension MOV has a change rate of 0.00 since no MOV was involved in a git commit. A change rate of 0.00 is not realistic, since a bit level error always may happen.

To calculate the efficiency of the two-stage-hierarchical strategy, I have to sum up every changed file in the dataset which results in 510 changed files in the dataset. The change rate of the dataset is 510/1560, which results in 0.32. The input for Equation 6.5 is therefore N=1560; p=0.32 and k=3.0. With context-sensitive pooling, another increase

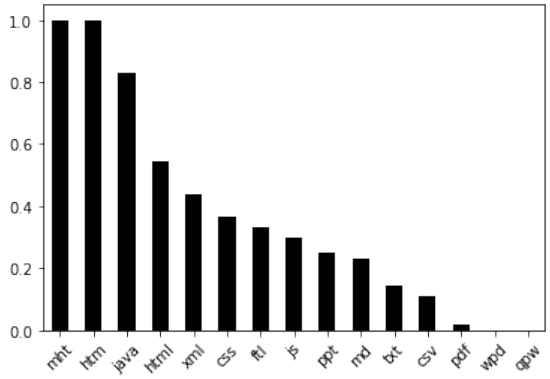


Figure 6.4: Distribution of change rates p.

extension	N	p	k	E(S_h)	W(S_h)
XML	986	0.43	2	1.21	2.00
PDF	106	0.01	8	3.60	7.57
MD	74	0.22	3	1.37	2.96
MOV	61	0.00	61	61.00	61.00
JAVA	47	0.82	2	1.21	1.95
None	26	0.00	26.0	26.00	26.00
ZIP	17	0.00	17	17.00	17.00
TXT	14	0.14	4	1.33	3.50
DOC	13	0.00	13	13.00	13.00
JP2	12	0.00	12	12.00	12.00
CSS	11	0.36	2	1.07	1.83
HTML	11	0.54	2	1.06	1.83
JS	10	0.3	3	1.08	2.50

Table 6.2: In group efficiencies of various file extensions

Strategy	E(S)	C(S)
Individual	1.00	1.00
Two stage hierarchical	1.28	3.0
Context Sensitive	1.38	2.35

Table 6.3: Efficiency of context-sensitive pooling vs. two stage hierarchical pooling.

in efficiency can be made. In Table 6.3 I compare the two strategies presented before, with respect to Equation 6.2, where the Two-Stage-hierarchical pooling algorithm has achieved an efficiency rate of 1.28 and context-sensitive pooling 1.38[9].

Given that metadata has a higher change rate, what effect has the split of metadata and

[9] `https://github.com/metsch/masterthesis/blob/main/src/py/rq2.ipynb`

Extension	N
METADATA	1560
XML	986
PDF	106
MD	74
MOV	47

Table 6.4: Arbitrary change rates p in the format-corpus dataset

p	p_{meta}	E(S_{cs})	C(S_{cs})	E(S_h)	C(S_h)
0.001	0.99	0.77	1.71	0.86	1.91
0.007	0.96	0.73	1.60	0.80	1.76
0.013	0.94	0.71	1.55	0.78	1.69
0.019	0.91	0.69	1.49	0.76	1.62
0.026	0.89	0.68	1.46	0.74	1.58
0.032	0.86	0.67	1.46	0.73	1.58
0.038	0.84	0.67	1.42	0.72	1.52
0.044	0.81	0.66	1.42	0.71	1.52

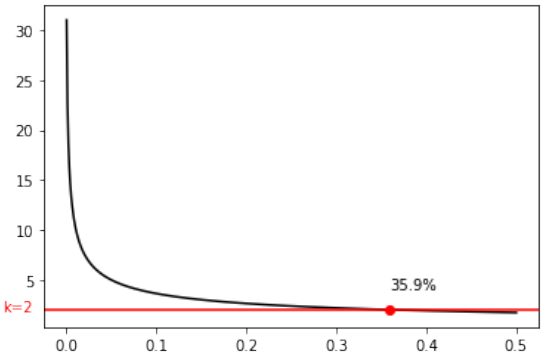
Table 6.5: Various change rates p and p_{meta} in the format-corpus dataset compared with the individual testing strategy S_h where 1560 cost relevant transactions and 2070 operations are needed.

objects on the operation cost? To know what effect the split off metadata has, I had to double the initial 1560 file extensions and assign the postfix .meta to the newly created rows resulting in Table 6.4, I calculate the efficiency for context-sensitive and two-stage hierarchical for various combinations of p and p_{meta} and the result is resulting in Table 6.5. The combinations for p and p_{meta} are set arbitrary, future work may include the estimation of change rates based on the file sizes. The Context-sensitive strategy never even reaches a 1.00 efficiency, which would state that it performs worse as the individual testing in terms of efficiency, this is due to the implementation of the context-sensitive strategy where the pool sizes cannot exceed the group size cannot exceed and loose the advantage of low change rates. In the two-stage-hierarchical I can group the metadata independently of the objects, resulting in 1560 objects with high change rate and 1560 objects with very low change rate. Although the performance is better than individual testing in terms of cost, the split off-metadata does have a worse effect on the overall performance due to the double amount of objects that have to be preserved[10].

The goal of this thesis was to reduce the cost of the fixity storage by at least 50%, this goal can be met when the optimal pool size is at least two, which is guaranteed with a prevalence lower than 35.9%, as seen in Figure 6.5. What happens if the prevalence is higher than 35.9%, the optimal pool size would be between 1.0 and 2.0 which is not practical since a pool can not include 1.42 objects, therefore I have decided to round the

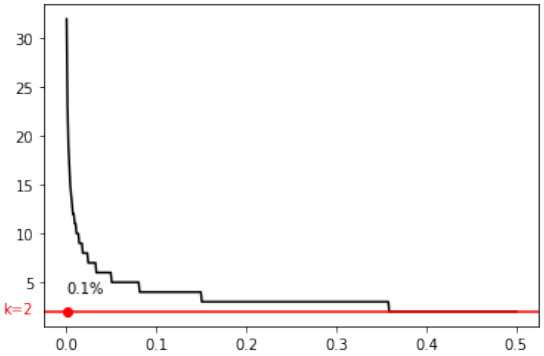
[10] https://github.com/metsch/masterthesis/blob/main/src/py/rq3.ipynb

Figure 6.5: For a prevalence of corruption of maximum 35.9%, an optimal pool size of at least two is guaranteed



result up from Equation 6.6 which results in Figure 6.6.

Figure 6.6: For a prevalence of corruption of maximum 35.9%, an optimal pool size of at least two is guaranteed



CHAPTER 7

Discussion

I have evaluated two distinct pooling strategies on an ingest consisting of 1560 digital objects. The change rate of the dataset is 32%, meaning that a third of the objects are expected to be corrupted or changed during the storage in the archive. With an individual testing strategy, 1560 writing operations on the blockchain are needed. The exact gas cost, to persist 1560 SHA256 values is $1560 \cdot 42,368 = 66,094,080$. The cost in USD for that much gas is \$13218.81, assuming a gas price of 50 Gwei and an Ether price of 4000, see Equation 5.2. It is important to denote the operation cost in gas, because the gas price is changing in the minute ranging from 30 Gwei up to 60 Gwei and the price of the Ether also fluctuates. With a gas price of 30 Gwei, the operation cost for an ingest of 1560 objects would decrease to \$7931. Therefore, it is important to be aware of the changing gas prices throughout the day. Gas prices are an indicator on how active the Ethereum network is at the moment, e.g. the gas prices are the lowest in the early morning in Europe when North America is mostly asleep. I have shown, that the amount of costly writing operations can be decreased up to three times utilizing pooled testing. With a two-stage-hierarchical pooling strategy, the amount of writing operations is 520, which results in a total gas cost of $22,031,360$ (\$4406.27). Although the operation cost was decreased to a third, the cost is still high for a digital archive considering that only a small part of the preservation process is done, persisting fixity information. An operator has to decide what is more important to them, the cost, the security or the convenience of their chosen storage-medium for fixity information.

My proposed implementation of pooled testing, utilizing hash-lists, does not have any method to find out which exact digital object in the pool has changed. On retrieval of a changed object, one has to get the SHA256 value from the blockchain and re-compute the SHA256 value of a pool with its respective digital objects. If a mismatch occurs of the two cryptographic hashes, there is no way of defining which exact object in the pool has changed. This is due to the fact, that the information on the integrity of single objects is lost while computing the root hash of a hash-list. Only the resulting root

hash matters in order to guarantee the integrity of the whole pool. The downside of this method is, that one has to replace the whole pool with correct copies of the objects every time a corrupt pool is detected. In order to lower the amount of these data-scrubbing operations, smaller pool sizes are considered in the proposed pooled testing strategies. In my thesis, I considered bit level error and data manipulation as possible sources of data change. For legitimate updates, one has to either persist an old version of the object or only persist the SHA256 of the object in order to be able to re-compute the root hash of the original pool. The updated object than have to be pooled in a new pool with other objects or individually and its hash re-uploaded to the Ethereum blockchain.

The first research questions was to find the optimal pool size for the fixity information with respect to two metrics: the first was to minimize the cost and the second was to maximize the efficiency. The two of them were inversely correlated, where to minimize the cost the pool size was actually N, since there was only one writing transaction on the blockchain when we have only one pool on ingest. On the other hand, a pool size of N is extremely inefficient on retrieval. On retrieval, I have to replace the whole ingest bulk with correct copies if only one object in the pool is corrupt, since I cannot determine which object in the pool got corrupted due to the nature of hash lists. Therefore, I had to also account the number of data scrubbing operations into the optimal pool size, resulting in Equation 6.4. The pool size is therefore heavily dependent on the change in the ingest bulk, where if the change rate is high smaller pool sizes are favorable. For the second research question, I have compared two different approaches for pooled testing. Where the efficiency of both strategies is compared to the efficiency of individual testing. The efficiency of individual testing for sample population of 100 objects where 20 of them are corrupt, meaning that I need 100 writing and additionally 20 data-scrubbing operations during the preservation process. With two stage hierarchical pooling an efficiency of 1.28 can be achieved. In the format-corpus dataset there were just a few file extensions of 90 that were altered and had a change rate greater than 0.0, suggesting that the two-stage strategy was not optimal, since non-volatile files would give been grouped together creating unnecessary traffic on repairing a corrupt pool. Therefore, I have proposed a context-sensitive approach where the inner group change rate of each file extension is taken into account. Each group in the ingest-bulk is assigned to an own pool with their respective pool size. For instance, in table 6.2 the file extension PDF has 8 corrupted objects out of 106, which results in a pool size of 8, whereas the XML extension has a much higher corruption rate with an optimal poolside of 2. The context-sensitive approach takes these differences into account and pools homogenous groups, where groups with 0.00 change rate can be grouped with a pool size as large as the inner group. With this method, groups with low change rate can be pooled with large poolside and therefore reduce the cost and increase the efficiency even more than individual testing or two-stage hierarchical pooling. In the third research question, where I tried to split-off metadata from the digital objects in order to create a bulk of highly volatile objects and a bulk of stable objects with low change rate, I showed that the approach is worse than with non split-off metadata. This is due to the fact, that I have to persist the double amount of digital objects and the pooling strategies could not

compensate for the high amount of objects.

The limitations of the strategies proposed in this thesis is the missing data on change rates on digital objects, this is due to the fact that I would have had to monitor an archive and measure the amount of times an object has changed. In this thesis, I have arbitrarily chosen change rates for the objects and their metadata, see Table 6.5. Future work may propose a method on how to estimate the change rate of an ingest in order to calculate the real optimal pool size.

The results indicate that the cost for the decentralized fixity storage on the Ethereum blockchain can be reduced, depending on the change rate of the digital objects, by at least 50%.

CHAPTER 8

Conclusion

While the Ethereum blockchain has its limitation regarding the cost, the results of pooled testing showed in this thesis may include the blockchain in future implementations of fixity information storages. This research has shown that the operation cost can be reduced up to a third of cost, but also raises the question if that is enough to establish the Ethereum blockchain as a storage medium for fixity information due to its high cost. While the security and convenience of the Ethereum blockchain stands out, the cost of persisting fixity information for large ingest bulks in digital archives is still high, even after reducing the cost with pooled testing. Operators of digital archives may consider only persisting the fixity of hand-picked digital objects, where security and availability is needed the most, on the blockchain.

A digital archive could deploy the fixity storage on the Ethereum network by setting up an own Ethereum node or by utilizing the developer tools presented in this thesis. After deployment, the operator of the digital archive would be immediately able to persist fixity information on the blockchain after the installation of an Ethereum client of their choice, such as Pythons Web3.py. Due to its minimal requirements, the proposed solution is lightweight and does not require a larger infrastructure to be executed, since the infrastructure to store the fixity information is provided by the Ethereum blockchain.

The optimal pool size for an ingest bulk is dependent on the change rate of its object. For an ingest of N digital objects, the optimal pool size regarding the operation cost is N. With a pool of size N, you only have to write once on the blockchain and therefore minimize the cost. But, in order for a pooling strategy to be efficient, smaller pool sizes have to be chosen to minimize the amount of data-scrubbing operations. Therefore, pool sizes from 2 to 10 are favorable, depending on the change rate of the objects. A pool with size 10 is already at the limit of efficiency and can only be considered for non-volatile digital objects which are not expected to change and may only suffer from bit level errors.

With pooled testing, I was able to reduce the operation cost of preserving the fixity information of 1560 digital objects up to a third. The dataset used is the OpenPreserve format-corpus, an openly licensed dataset consisting of various file set and files from creation tools. The cost, utilizing an individual testing strategy was $13218.81 whereas with a two-stage-hierarchical approach the cost was $4406.27. In my thesis, I have also shown that the operation cost in USD can not be considered to be relevant, since it is heavily dependent on the price of gas and the Ether token, the native cryptocurrency of the Ethereum blockchain. In my thesis, I have chosen to present the operation cost in gas which is a constant value and does not change even when the Ether price rises or falls. The presented amount of gas can be converted into USD or any other currency. I have also shown, that the efficiency can be improved by a factor of 1.38 with the implementation of context-sensitive pooling. The context-sensitive approach is slightly more efficient than the two-stage-hierarchical pooling due to its consideration of inner group change rates, whereas two-stage-hierarchical pooling only considers a single change rate over the whole ingest bulk. The third approach, utilizing split-off metadata has shown to have a worse the cost and efficiency compared to with non split-off metadata. Although, considering with split-off metadata you have the double amount of digital objects, the cost of this approach is still almost halved compared to individual testing.

Future work may include the migration of my proposed fixity information storage to another blockchain, a cheaper one. Although you have to consider that the price on the established Ethereum network has its security advantages, e.g. denial-of-service attacks only hurts the attacker because of the cost. A good consideration are so-called Layer 2 solutions, which is a term for Ethereum scaling solutions that handle transactions off Ethereum Layer 1 while still taking advantage of the security[1]. Layer 2 solutions are considered faster and cheaper and therefore would be great candidate to enhance the proposed fixity information storage.
Future work may also include the estimation of change rates in ingest-bulks in order to compute the optimal pool size. In my work, I estimated the change rate of the format-corpus dataset by counting the involvement of certain file extensions in Git commits. My suggestion for the estimation is to monitor a digital archive for a certain amount of time and count the amount of times a certain object has changed. Another interesting method would be, to estimate the change rate of an object based on its file size. Assuming that a larger file has a higher chance to experience a bit level error during long term storage.

My contribution to archival science and preservation research was to gather the exact computational effort, in form of gas, needed to operate a decentralized fixity information storage on the Ethereum blockchain. Additionally, I have presented a method to decrease the amount of operations and therefore the amount of cost relevant transactions needed to operate such an application. With pooled testing, the operation cost of storing the fixity information for 1560 files in format-corpus dataset has been reduced to a third of the cost.

[1] `https://ethereum.org/en/layer-2/`

List of Figures

List of Tables

Glossary

call An Ethereum call refers to an interaction with a smart contract which does not change the state on the blockchain..

data-scrubbing Data-scrubbing is an error correction technique which replaces corrupted object with copies..

history-forgery The creation of false artifacts..

transaction An Ethereum transaction refers to an action initiated by an account managed from a human, changes the state of the blockchain..

Acronyms

S_{cs} context-sensitive pooling strategy.

S_h Two-stage-hierarchical pooling strategy.

S_i Individual testing strategy.

C(S) Cost Efficiency of strategy S compared to individual testing.

C(S) Cost-Efficiency of strategy S compared to individual testing.

DAPP Decentralized Application.

DLT Distributed Ledger Technology.

E(S) Efficiency of strategy S compared to individual testing.

ETH Ether.

EVM Ethereum Virtual Machine.

GWEI Gwei.

k Optimal Pool Size.

O(S) Number of operations needed with strategy S.

OAIS Open Archival Information System.

p Prevalence of corruption.

R(S) Number of data scrubbing operations needed with strategy S.

T(S) Operation Cost of strategy S in gas.

W(S) Number of cost relevant operations needed with strategy S.

Bibliography

[CBB+18] John Collomosse, Tu Bui, Alan Brown, John Sheridan, Alex Green, Mark Bell, Jamie Fawcett, Jez Higgins, and Olivier Thereaux. Archangel: Trusted archives of digital public documents. In *Proceedings of the ACM Symposium on Document Engineering 2018*, pages 1–4, 2018.

[CGWK20] Alhaji Cherif, Nadja Grobe, Xiaoling Wang, and Peter Kotanko. Simulation of pool testing to identify patients with coronavirus disease 2019 under conditions of limited test availability. *Jama network open*, 3(6):e2013075–e2013075, 2020.

[Dan17] Chris Dannen. *Introducing Ethereum and solidity*, volume 1. Springer, 2017.

[DBK20] Andreas Deckert, Till Bärnighausen, and Nicholas NA Kyei. Simulation of pooled-sample analysis strategies for COVID-19 mass testing. *Bulletin of the World Health Organization*, 98(9):590, 2020.

[DFG+14] Paula De Stefano, Carl Fleischhauer, Andrea Goethals, Michael Kjörling, Nick Krabbenhoeft, Chris Lacinak, Jane Mandelbaum, Kevin McCarthy, Kate Murray, Vivek Navale, and Others. Checking Your Digital Content: What is Fixity, and When Should I be Checking It? *National Digital Stewardship Alliance*, 2014.

[Dor43] Robert Dorfman. The detection of defective members of large populations. *The Annals of Mathematical Statistics*, 14(4):436–440, 1943.

[Dur09] Luciana Duranti. From digital diplomatics to digital records forensics. *Archivaria*, pages 39–66, 2009.

[Hev07] Alan R Hevner. A three cycle view of design science research. *Scandinavian journal of information systems*, 19(2):4, 2007.

[HS05] Mark Hofer and Kathleen Owings Swan. Digital image manipulation: A compelling means to engage students in discussion of point of view and perspective. *Contemporary Issues in Technology and Teacher Education*, 5(3):290–299, 2005.

[KAN+17] K. Kim, D.S. Alliance, B. Nowviskie, W. Graham, B. Quon, C. Kussmann, W. Atkins, A. Reich, M. Schultz, and L. Work. Fixity survey report: An ndsa report. http://dx.doi.org/10.1.1/jpb001, 2017.

[KGA+14] K. Kim, W. Graham, D.S. Alliance, A Reich, and C. Kussmann. What is Fixity, and When Should I be Checking It. https://osf.io/an5zh/, 2014.

[KORD10] M.G. Kirschenbaum, R. Ovenden, G. Redwine, and R. Donahue. *Digital Forensics and Born-digital Content in Cultural Heritage Collections*. CLIR publication. Council on Library and Information Resources, 2010.

[Lee10] Christopher A Lee. Open archival information system (OAIS) reference model. *Encyclopedia of library and information Sciences*, 3, 2010.

[NEG+21] Roch A Nianogo, I Obi Emeruwa, Prabhu Gounder, Vladimir Manuel, Nathaniel W Anderson, Tony Kuo, Moira Inkelas, and Onyebuchi A Arah. Optimal uses of pooled testing for COVID-19 incorporating imperfect test performance and pool dilution effect: An application to congregate settings in Los Angeles county. *Journal of medical virology*, 93(9):5396–5404, 2021.

[REHHR20] Francesca Regen, Neriman Eren, Isabella Heuser, and Julian Hellmann-Regen. A simple approach to optimum pool size for pooled sars-cov-2 testing. *International Journal of Infectious Diseases*, 100:324–326, 2020.

[SBP+20] Marten Sigwart, Michael Borkowski, Marco Peise, Stefan Schulte, and Stefan Tai. A secure and extensible blockchain-based data provenance framework for the Internet of Things. *Personal and Ubiquitous Computing*, 2020.

[Woo14] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger berlin version. 2014.

[WY21] Hosung Wang and Dongmin Yang. Research and development of blockchain recordkeeping at the National Archives of Korea. *Computers*, 10(8):90, 2021.

[ŽLG21] Julius Žilinskas, Algirdas Lančinskas, and Mario R Guarracino. Pooled testing with replication as a mass testing strategy for the COVID-19 pandemics. *Scientific Reports*, 11(1):1–7, 2021.