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, 1. Jänner 2001

Michael Etschbacher

Andreas Rauber

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, 1st January, 2001

Michael Etschbacher

Andreas Rauber

Erklärung zur Verfassung der Arbeit

Bsc. Michael Etschbacher

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. Jänner 2001

Michael Etschbacher

Danksagung

Ihr Text hier.

Acknowledgements

Enter your text here.

Kurzfassung

Ihr Text hier.

Abstract

Enter your text here.

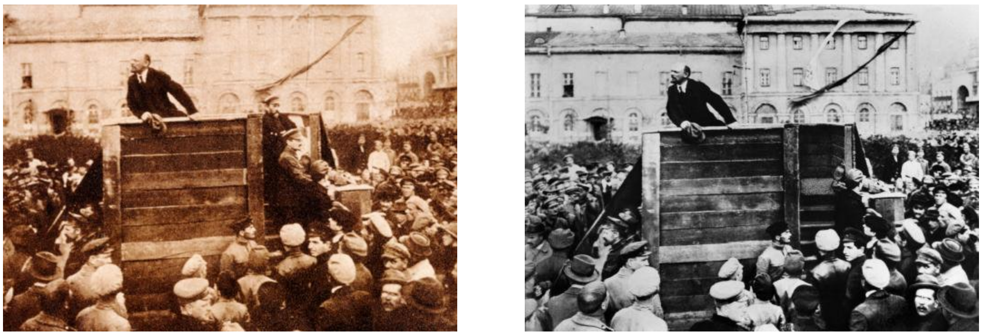
Contents

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 1.1, using paint, razors and airbrushes. Soviet photo artists altered the historical record by literally removing Trotsky from the pictures [HS05, 3].



(a) Original

(b) Original

Figure 1.1: Catalog Images

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, SHA-256) is relatively easy, managing that information over time is harder considering that if a malicious actor can alter the fixity information, the actor is also able to alter the underlying object illicit [KORD10, 35].Fixity information is usually stored in databases; object metadata records or alongside content, whereas this thesis deals with blockchain as a storage medium. [CBB$^+$18] and [SBP$^+$20] have shown that the Ethereum blockchain, implemented by [B$^+$13], can indeed be utilized to ensure data integrity, but there is a problem with that: the operation cost. The cost of storing a SHA256 values on the Ethereum blockchain is 20.000 gas (the unit for transaction fees), which oscillate at the time of writing at about 5$, which means the operation cost for an ingest of 10.000 objects costs about 50.000$.

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.

1.4 Research Questions

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

RQ1 What is the optimal pool size based on the corruption rates of digital objects in the archive in terms of efficiency and cost

RQ2 To what extend can pooled object hashes increaes the efficiency and reduce cost for a fixity information storage service on the ethereum blockchain?

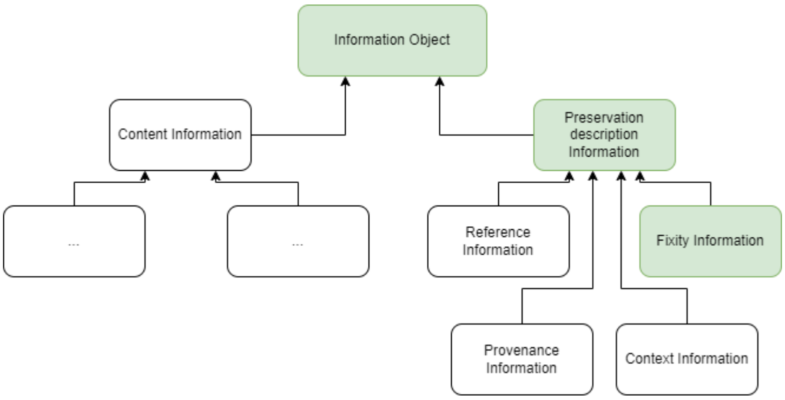
RQ3 Given that metadata has a higher corruption rate, what effect has the split of metadata and objects on the operation cost?

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 which manages any kind of data, data will be referenced as information object in the archive and each object is provided with necessary metadata, as described in the OAIS reference model. I assume that the data integrity is 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 infromation in the form of a unique transaction hash, see Figure 1.2.

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]



The *Artefact* is a combination of decentralized application on the ethereum blockchain and a local pool strucutre 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 tamper proof 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 artefact should reduce the amount of operations needed to ensure the integrity of 10.000 objects from ingest to retrieval process by at least 50%.

CHAPTER 2

Related Work

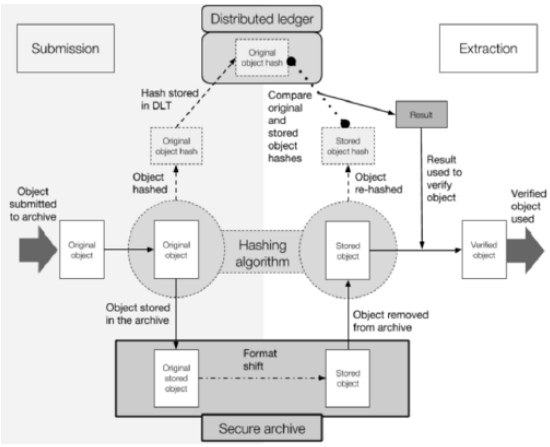
2.1 Project Archangel

Project ARCHANGEL was a de-centralised fixity information developed in 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 an 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 authors of the project ARCHANGEL stated that the trust conferred by the general public to the Archives and Memory Institutions (AMIs) had eroded due to the ease by which forgery and unauthorized modifications to electronic records were conducted owing to advances in technology and the generation of numerous types of composited content. To recover public trust in AMIs and guarantee the integrity of the records, a method utilizing the existing databases as well as a method to utilize a Merkle tree or durable storage provided by private corporations have been taken into consideration; however, it was concluded that they are difficult to apply owning to several limitations. Unlike in the past, when archives have relied on the products and technologies of certain companies, blockchain presents a completely different paradigm from opennes and expandability. The blockchain awards permission to write records in a distributed ledger to only authorized institutions, whereas permission to view the revorded content after accessing the distributed ledger is granted to every node participating in the blockchain. Moreover, scalability allows the utilization of various tools provided by open-source codes and enables the integrity of records to be assured by multiple parties through a consensus mechanism rather than by a single centralized institution [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 than 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 hashvalue are ident. Within the ARCHANGEL project, a combination of blockchain and artificial intelligence was attempted in a bid to ensure the integrity of audiovisual records. Audiovisual records require a continuous conversion of formats for utilization, and the converted record has a different hash value even with the same content as the original copy if the format is changed. To resolve this problem, the ARCHANGEL project developed the temporal content hash by applying the deep neural network as a kind of artificial intelligence technique [BCC+19, 3].

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



In this research, the machine learning algorithm was executed with the aim of extracting the content characteristics of audiovisual records irrespective of whether the format was converted. As the unique value is extracted by capturing the content characteristics regardless of the format conversion, TCH was the concept suggested for the first time by the ARCHANGEL project [BCC+19, 3]. This research proposed the application of TCH for the replacement of SHA 256 hash values as a means to guarantee the integrity of audiovisual records [WY21, 4].

2.2 Provenance framework for the IOT

[SBP+20] proposed a generic data provenance framework for the IoT. Their approach is to store provenance records on the Ethereum blockchain. Their approach is to store provenance records on the blockchain, with the goal of immutable provenance records for supply chains or other use cases [SBP+20, 4].

The framework is interesting for my thesis, since the framework can be adapted to store fixity information instead of provenance data by simply adapting the smart contract code of the project, which is publicy available on github [1].

2.3 Remote Data Integrity Checking Scheme

[ZCL$^+$20] proposed a blockchain based remote data integrity checking scheme. Their scheme consists of three phases, including setup, storage and verification phase, this scheme may be of great importance for my work because it integrates well with the work of Sigwart et al. 2020 in terms of design goals. The detail of each phase in their scheme is described as follows: (1) Setup phase, where the public parameter of the elliptic curve group is uploaded to the blockchain. (2) Storage phase, where the data owner signs the data blocks and uploads them to a cloud storage. (3) Verification phase, where the data owner downloads the data or desires to check the integrity of the individual data. The auditing process is done by verifying the signature of the data blocks by the data owner.

[1] `https://github.com/msigwart/iotprovenance`

CHAPTER 3

Diplomatics of Digital Records

3.1 About Diplomatics

Diplomatics is a science that was developed in iplomatics is a science that was developed in France in the seventeenth century by the Benedictine monk Dom Jean Mabillon in a treatise Diplomatics is a science that was developed in France in the seventeenth century by the Benedictine monk Dom Jean Mabillon in a treatise entitled De Re Diplomatica Libri VI (1681) for the purpose of ascertaining the provenance and authenticity of records that attested to patrimonial rights. It later grewinto a legal, historical, and philological discipline as itcame to be used by lawyers to resolve disputes, by historians to interpret records, and by editors to publish medieval deeds and charters. It is useful to distinguish classic diplomatics from modern/digital diplomatics, because these two branches of the discipline do not represent a natural evolution of the latter from the former, but exist in parallel and focus on different objects of study. Modern diplomatics has a broader scope; it is concerned with all documents that are created in the course of affairs of any kind, and is define as "the discipline which studies the genesis, forms, and transmission" of records, and "their relationship with the facts represented in them and with their creator, in order to identify, evaluate, and communicate their true nature". The primary focus of both classic and modern diplomatics is to assess the trustworthiness of records; however, the former establishes it retrospectively, looking at records issued several centuries ago, while the latter is concerned not only with establishing the trustworthiness of existing records but also with ensuring the trustworthiness of records that have yet to be created. [KORD10, 10].

Digital diplomatics defines a digital record as a digital component, or group of digital components, that is saved, and 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. A digital record is distinguished from an analogue record and an electronic record. InterPARES considers analogue the representation of an object or physical process through the use of continuously variable electronic signals or mechanical patterns. In contrast to a digitally encoded representa tion of an object or physical process, an analogue representation resembles the original. InterPARES defines an electronic record as any analogue or digital record that is carried by an electrical conductor and requires the use of electronic equipment to be intelligible by a person. InterPARES defines a record, using the traditional archival concept, as a document made or received in the course of a practical activity as an instrument or a by-prod uct of such activity, and set aside for action or reference [Dur09, 52].

The integrity of a record is inferred not only from its appearance – which might be deceiving in the case of good forgeries – but also from the circumstances of its maintenance and preservation: an unbroken chain of responsible and legitimate custody is considered an insurance of integrity until proof to the contrary, and integrity metadata are required to attest to that. The authenticity of a record is a movable responsibility, as it shifts from the creators trusted recordkeeper, who needs to guarantee it for as long as the record is in its custody, to the trusted custodian, who guarantees it for as long as the record exists. [Dur09, 53]. In my thesis the Ethereum network acts as a trusted custodian which guarantees that no one is able to tamper with the cryptographic hash values of digital records.

3.2 Trustworthiness

Trustworthiness is a concept and an obligation that spans the life of a document, whether it is a sheaf of paper or a WordPerfect file. The needs of born-digital objects shift as files move through the stages of the preservation process, from initial capture and metadata extraction to longer-term strategies such as migration and rights management. Born-digital fonds are similarly mobile as they pass from the creator, to an intermediary such as a dealer or other agent (human or technological), to staff at an archival repository, and, finally, to storage and, perhaps, ingest into a digital repository. The stages of that journey constitute the chain of custody for a digital object, and each stage has important implications for the trustworthiness of the born-digital materials in a given accession. Authentic source may be deceptive or unreliable, and although reliability is an important component of trustworthiness, the veracity of a documents content is often not the concern of archivists working with cultural heritage materials. Rather, the provenance of both analog and digital materials, as well as documentation about their storage environment, what has been done to them, and by whom, are the key aspects of establishing and maintaining trust. Trustworthiness— of an institution, a custodian, or a document—plays an important role in the acquisition and maintenance of born-digital materials. How best to determine and document that quality in a digital environment and with regard to the stewardship of born-digital materials is a question that remains under consideration [KORD10, 27].

Digital diplomatics concerns itself with five aspects of trustworthiness: reliability, authenticity, accuracy, integrity and authentication [KORD10, 10].

3.2.1 Reliability

Reliability is the trustworthiness of a record as a statement of fact, as to content. It is assessed on the basis of 1) the completeness of the record, that is, the presence of all the formal elements required by the juridical-administrative system for that specific record to be capable of achieving the purposes for which it was generated; and 2) the controls exercised on the process of creation of the record, among which are included those exercised on the author of the record, who must be the person competent, that is, having the authority and the capacity, to issue it. The reliability of a record is the exclusive responsibility of its creator and the trusted recordkeeper, that is, of the person or organization that made or received it and maintained it with its other records [Dur09, 52].

The concept of reliability, used in reference to the source of the records, is defined in digital forensics in a way that points to a reliable software, measured by principles similar to those the courts use to determine evidentiary reliability, that is, empirical testing, subjection to peer review and publication, determination of error rate, and general acceptance within the relevant community. Also these principles point to open source software because the processes of records creation and maintenance can be authenticated with evidence either by describing a process or system used to produce a result, or by showing that the process or system produces an accurate result [Dur09, 59].

3.2.2 Authenticity

Authenticity is the trustworthiness of a record as a record, and is defined as the fact that a record has not been tampered with or corrupted, either accident ally or maliciously. An authentic record is one that preserves the same identity it had when first generated, and can be presumed or proven to have maintained its integrity over time. The identity of a record is constituted of the whole of those characteristics that distinguish it from any other record, and is assessed on the basis of the formal elements on the face of the record, and/or its attributes, as expressed for example in a register entry or as metadata [Dur09, 52].

The expectations of scholars with regard to the reliability of sources have evolved over the centuries, from the assumption that librarians and archivists would present researchers with evidence that could be relied upon to be verifiable, to more modern understandings that dispense with the ideal of the reliable source and consider all texts as potentially deceptive and richly ambiguous. Ideally, the methods of operation and processes developed by repositories over years of working with scholars and other patrons enable staff to provide researchers with documentation about the provenance and acquisition of the items in their care [KORD10, 32].

3.2.3 Accuracy

Authenticity is the trustworthiness of a record as a record, and is defined as the fact that a record has not been tampered with or corrupted, either accident ally or maliciously. An authentic record is one that preserves the same identity it had when first generated, and can be presumed or proven to have maintained its integrity over time. The identity of a record is constituted of the whole of those characteristics that distinguish it from any other record, and is assessed on the basis of the formal elements on the face of the record, and/or its attributes, as expressed for example in a register entry or as metadata [Dur09, 52].

3.2.4 Integrity

In the digital environment, there are no originals in the diplomatics sense, that is, there are no records which, in addition to being complete and capable of reaching the purposes for which they were generated (i.e., effective) are also the first instance of each item under consideration, because when we close a digital record for the first time we destroy the original and every time we open it we create a copy. However, we can state that each digital record, in the last version used by the creator in the usual and ordinary course of business, is a copy in the form of original and, in any version kept by the preserver, is an authentic copy of the record of the creator. They are both authoritative and authentic if their iden tity is intact and their integrity can be either presumed or proven. When extrating digital evidence, digital forensics must, first of all, avoid altering the data, and are guaranteed reliable in such sense by ensur ing that they are repeatable. Repeatability, which is one of the fundamental precepts of digital forensics practice, is supported by the accurate documentation of each and every action carried out on the evidence [Dur09, 58]. Such a chain of alteration is possible on the Ethereum network, since each update on an objects hash is natively documented on the blockchain. Duplication integrity is ensured when given a data set, the process of creating a duplicate of the data does not modify the data (either intentionally or accidentally) and the duplicate is an exact bit copy of the original data set. It is possible to preserve data integrity over the duplicate, with respect to the original, by using a trusted third party. At the time the image is created, a copy of the hash can be given to a trusted third party to hold in escrow. Now changes to the duplicate can be detected even if the original is modified. Digital forensics experts also link duplication integrity to time and have considered the use of time stamps for that purpose.63 A distinction between the integrity of a record as such and that of its duplicate may be useful to eliminate the conflict between the view of integrity held by diplomatists and that held by information technology experts, who tend to support the need for the extreme authentication provided by a digital signature. Indeed, one could further enrich the concept of integrity by also adopting the link between integrity and time proposed by digital forensics experts, and define record integrity differently in each phase of the record life cycle and/or custodial history [Dur09, 60]. Both diplomatics and forensics were developed as practices for the purpose of investigating existing material evidence, assessing its status of transmission, its authenticity, and its ability to provide proof of facts at issue [Dur09, 64].

3.2.5 Authentication

Authentication is defined as a declaration of authenticity made by a competent officer, and consists of a statement or an element, such as a seal, a stamp, or a symbol, added to the record after its completion. While authenticity is a quality of the record that accompanies it for as long as it exists as is, authentication only guarantees that a record is authentic at one specific moment in time, when the declaration is made or the authenticating element or entity is affixed. In the digital environment, extreme authentication is usually provided by a igital signature. The digital signature has the function of a seal because it is attached to a complete record, allows verification of the origin and integrity of the record, and makes the record indisputable and incontestable by performing a non-repudiation function [Dur09, 53].

CHAPTER 4

Ethereum Blockchain

4.1 Introduction to Ethereum

Open source blockchain networks such as Ethereum and Bitcoin are kits that allow you to set up an economic system in software, complete with account management and a native unit of exchange to pass between accounts. These native units of exchange are called coins, tokens, or cryptocurrencies, but the are no different from tokens in any other system: they are a form of money that is usable only within that system to simply pay your peers or to 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 Web3[1],as utilized in this thesis to connect to the Ethereum network [Dan17, 2]. The fact that everyone in the world can interact with the same distributed database, given you have internet access and a decent device is favoritable for a fixity storage such as presented in 6 since anyone will be able to confirm if an object has changed over time if given the right resources, such as the transaction hash in which the fixity information of an object was stored on the blockchain. Lets picture an digital archive which releases multiple objects to the public and their respective fixity information on the blockchain, citizens 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 blockchain

A block is a unit of time that encompasses a certain number of transactions. Inside that period, transaction data is recorded; when the unit of time elapses, the next block begins. The blockchain represents the history of state changes within the network database of

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

the EVM [Dan17, 43]. Transactions and state changes in the Ethereum network are segmented into blocks, and then hashed. Each block is verified and validated before the next canonical block can be placed on top of it. In this way, nodes on the network do not need to individually evaluate the trustworthiness of every single block in the history of the Ethereum network, simply to compute the present balances of the accounts on the network. They merely verify that its “parent block” is the most recent canonical block. They do this quickly by looking to see that the new block contains the correct hash of its parents transactions and state. All the blocks strung together, and including the genesis block, an honorific describing the first block the network mined after coming online, are called the blockchain. In some circles, you will hear the blockchain referred to as a distributed ledger or distributed ledger technology (DLT). Ledger is an accurate description, as the chain contains every transaction in the history of the network, making it effectively a giant, balanced book of accounts [Dan17, 55].

4.3 Ethereum Virtual Machine

The EVMs physical instantiation can not be described in the same way that one might point to a cloud or an ocean wave, but it does exist as one single entity maintained by thousands of connected computers running an Ethereum client [2]. A virtual machine (VM), in the Ethereum context, is one giant global computer composed of constituent nodes, which are themselves computers too. Generally speaking, a virtual machine is an emulation of a computer system by another computer system. These emulations are based on the same computer architectures as the target of their emulation, but they are usually reproducing that architecture on different hardware than it may have been intended for. Virtual machines can be created with hardware, software, or both. In the case of Ethereum, it is both. Rather than securely network thousands of discrete machines, Ethereum takes the approach of securely operating one very large state machine that can encompass the whole Earth [Dan17, 48]. The EVM can run arbitrary computer programs written in the Solidity[3] language. These programs, given a particular input, will always produce the output the same way, with the same underlying state changes. This makes Solidity programs fully deterministic and guaranteed to execute, provided youve paid enough for the transaction. Solidity programs are capable of expressing all tasks accomplishable by computers, making them theoretically Turing complete. That means that the entire distributed network, every node, performs every program executed on the platform [Dan17, 50]. From the perspective of a software developer, the EVM is also a runtime environment for small programs that can be executed by the network. The EVM has its own language, the EVM bytecode, to which smart contracts compile. Solidity, which is a high-level language, is compiled into bytecode and uploaded onto the Ethereum blockchain by using a client application such as truffle[4] or geth[5] [Dan17, 51].

[2] https://ethereum.org/en/developers/docs/evm/

[3] https://github.com/ethereum/solidity

[4] https://trufflesuite.com/

[5] https://geth.ethereum.org/

I have decided to use truffle for my thesis because of its fast and easy way to interact with the Ethereum network and its subnetworks.

4.4 Smart Contracts

Smart contracts are the building blocks of decentralized applications running on the EVM, they are like the concept of classes in conventional object-oriented programming. When developers speak of writing smart contracts, they are typically referring to the practice of writing code in the Solidity language to be executed on the Ethereum network. When the code is executed, units of value may be transferred as easily as data [Dan17, 10]. In my thesis, the fixity storage is often referred to as decentralized application (DAPP) which is built with several smart contracts. The smart contract implement in this thesis is further described in Section 6.3.

4.5 Persistence

Unlike a centralized server operated by a single company or organization, decentralized storage systems consist of a peer-to-peer network of user-operators who hold a portion of the overall data, creating a resilient file storage sharing system. These can be in a blockchain-based application or any peer-to-peer-based network. Ethereum itself can be used as a decentralized storage system, and it is when it comes to code storage in all the smart contracts. However, when it comes to large amounts of data, that isn't what Ethereum was designed for. 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 wouldn't be feasible for all nodes to continue to run. Also, the cost of deploying this much data to Mainnet would be prohibitively expensive due to gas fees [7]. My proposed solution utilizing pooled testing, presented in Chapter 5, mitigating the ever increasing chain size of the Ethereum network by reducing the amount of transactions needed to operate the fixity storage.The gas cost of storing a SHA256 value on the blockchain can be read from Table 4.8. TODO wie viel gas kostet ein sha256 wort

All transactions in Ethereum are stored on the blockchain, a canonical history of state changes stored on every single Ethereum node [Dan17, 12]. Like all databases, a blockchain has a schema: rules define, constrain, and enforce relationships between entities. Motivations to break or alter these relationships can be found across industries, leading to bribery and corruption, and making blockchains trustless qualities even more attractive to business than prior generations of software and networking. In all databases, shared read/write access creates enormous complexity. Machines all over the world may experience varying latency, depending on where the database is physically located, leading to some write operations arriving out of order. This gets even more difficult if several

[6] https://ycharts.com/indicators/ethereum_chain_full_sync_data_size

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

parties are supposed to equally share a database [Dan17, 20]. The nodes go through the block they are process and run any code enclosed within the transactions. Each node does this independently; it is not only highly parallelized, but highly redundant. Despite the high redundany, this is an efficient way to balance a global ledger in a trustworthy way [Dan17, 50].

4.5.1 Blockchain-based persistence

This type of persistence is utilized in my thesis, since the fixity information of the archived objects are meant to be persisted for longterm. For a piece of data to persist forever, Ethereum needs to use a persistence mechanism. For example the persistence mechanism is that the whole chain needs to be accounted for when running a node. New pieces of data get tacked onto the end of the chain, and it continues to grow - requiring every node to replicate all the embedded data. This is known as blockchain-based persistence. The issue with blockchain-based persistence is that the chain could get far too big to upkeep and store all the data feasibly (e.g. many sources estimate the Internet to require over 40 Zetabytes of storage capacity). The blockchain must also have some type of incentive structure. For blockchain-based persistence, there is a payment made to the miner. When the data is added to the chain, the nodes are paid to add the data on [8].

4.5.2 Contract-based persistence

Contract-based persistence has the intuition that data cannot be replicated by every node and stored forever, and instead must be upkept with contract agreements. These are agreements made with multiple nodes that have promised to hold a piece of data for a period of time. They must be refunded or renewed whenever they run out to keep the data persisted. In most cases, instead of storing all data on-chain, the hash of where the data is located on a chain gets stored. This way, the entire chain doesn't need to scale to keep all of the data [9].

4.6 Test Networks

. These networks are not really free of charge, but the ETH token on the respective network is free to get. Although the queue of the free ETH token can be very vast in some cases, e.g. `https://faucet.dimensions.network/` this faucet has a waiting queue for over three days until you recieve one ETH token for the ropsten network. Other faucets, a distributer of free ETH test tokens, require social network accounts to verifiy that you are not a bot e.g. `https://faucet.paradigm.xyz/` requires a twitter account with at least one tweet and at least 15 retweets before you can request one ETH token. The faucet request are almost all time gated, meaning that you could request ETH only once per day. At small scale, this limitation is no problem for this

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

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

thesis because persisting around 10 pools on the blockchains require at max 0.002 ETH tokens. The one per day paradigm gets problemeatic for the experiment for the 10.000 objects, which costs about 20 ETH tokens, which means I would have to request tokens fors 20 days straight to perform one experiment on a real live environment such as the ropsten testnetwork. There are multiple options to choose from for a testnetwork, from the most prominent networks presented at the official ethereum documentation [10] only the ropsten 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 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 me 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 refers to the fee required to conduct a transaction on Ethereum successfully. Gas fees are paid in Ethereum's native currency, ETH (ETH). 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 ETH[11]. Miners are paid in ETH for mining, and also for running scripts on the network. The cost associated with electricity expenditure of servers running on the Ethereum network is one of the factors that gives ETH, 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 completing a block and getting paid [Dan17, 12]. Mining achieves the consensus required to make valid state changes, and the miners are paid for contributing to the consensus building. This is how ETH and bitcoin are created [Dan17, 57]. For every instruction the EVM executes, there must be a cost associated, to ensure the system isnt jammed up by useless spam contracts. Every time an instruction executes, an internal counter keeps track of the fees incurred, which are charged to the user. Each time the user initiates a transaction, that users wallet reserves a small portion to pay these fees [Dan17, 58]. The costs are the driving factor of my thesis where I try to make as less transactions as possible to run a fixity storage on the Ethereum network. The fees are dependent on the gas cost of a transaction, gas is a unit of work; it is not a subcurrency, and you can not hold or hoard it. It simply measures how much effort each step of a transaction will be, in computational terms. To be able to pay for gas costs, you simply need to add ETH to your account. You do not have to acquire it separately; there is no gas token. Every operation possible on the EVM has an associated gas cost. Gas costs ensure that computation time on the

[10] https://ethereum.org/en/developers/docs/networks/

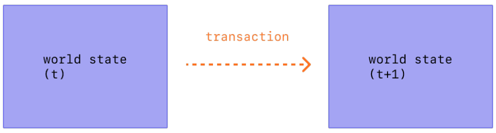
[11] https://ethereum.org/en/developers/docs/gas/

network is appropriately priced [Dan17, 59]. If you send a computationally difficult set of instructions to the EVM, the only person this hurts is you. The work will spend your ether, and stop when the ETH you allocated to the transaction runs out. It has no effect on anyone elses transactions. There is no way to jam up the EVM without paying a lot, in the form of transaction fees, to do it. Scaling is handled in a de facto way through the gas fee system. Miners are free to choose the transactions that pay the highest fee rates, and can also choose the block gas limit collectively. The gas limit determines how much computation can happen (and how much storage can be allocated) 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 ETH, Bob's account must be debited and Alice's must be credited. This state-changing action takes place within a transaction. Transactions, which change the state of the EVM, need to be broadcast to the whole network. Any node can broadcast a request for a transaction to be executed on the EVM; after this happens, a miner will execute the transaction and propagate the resulting state change to the rest of the network. [12]. Transactions, which do not alter the global state are free of charged and are referred to as Calls. In the source code, as seen in the source code in 6.1, *getPoolHash(uint32 poolId)* is marked with they 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 reffered to as calls.

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



TODO Verbatim von ana transaction einfügen, vl von ganache oder etherscan

Transactions come from external accounts, which are usually controlled by human users. It is a way for an external account to submit instructions to the EVM to perform some

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

operation. In other words, it is a way for an external account to get a message into the system. A transaction in the EVM is a cryptographically signed data package storing a message, which tells the EVM to transfer ether, create a new contract, trigger an existing one, or perform some calculation. Contract addresses can be the recipients of transactions, just like users with external accounts [Dan17, 60]. Table 4.8 presents the gas cost of the most relevant network operations used in this thesis according to the Ethereum yellow paper [Woo, 27].

Operation Name	Gas Cost	Description
$G_codedeposit$	200	gas cost per bytesize of the compiled bytecode of the smart contract
$G_txcreate$	32000	create a new smart contract
$G_transaction$	21000	retrieve the current balance of an account
$G_txdatanonzero$	16	gas cost per bytesize of the compiled bytecode of the transaction
$G_balance$	400	retrieve the current balance of an account

CHAPTER 5

Pooled Testing

5.1 Pooled Testing

Pooled Testing was first introduced by [Dor43] as a strategy to screen a large number of military recruits for syphiclis 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. Individuals from negative pools would be declared negative, and specimens from positive pools would be retested individually to identify which recruits had contracted the disease. Dorfmanss motivation for using group testing was to reduce testing costs while still identifying all syphilitic-positive recruits. Today, this would be described as the "case identification problem," because the goal is to identify all positive individuals among all individuals tested. Dorfmanss approach to case identification can be viewed as a two-stage hierarchical algorithm. In this approach, non-overlapping pools are tested in the first stage and individuals from positive pools are tested in the second stage. When the corruption prevalence is small, higher-stage algorithms have proven to be useful at further reducing the number of tests needed [HTBM17, 1].

5.2 Hierarchical pooling algorithms

Hierarchical algorithms involve testing samples in non-overlapping pools over a pre-defined number of stages, until each individual can be classified as positive or negative [LTM+21, 2].

5.3 Two Stage Hierarchical Pooling Algorithm

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 needied, but when a pool is declared positive a second stage is needed and all individuals from

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

Parameter	description
p	prevalence of corruption, the probability that an object will be corrupted during the preservation process
1-p	the probability that the integrity of an object is preserved during the preservation process
$(1-p)^n$	the probability of obtaining a uncorrupted result from a pool of n objects
$1-(1-p)^n$	the probability of obtaining a corrupted result from a pool of n objects
N/j	the number of pools of size j in a population of size N
p'N/j	the expected number of corupted pools of n in a poluation of N with a corruption rate of p
$E(T) = N/j + n(N/n)p'$	the expected number of tests

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]. The optimal pool size is the size that minimizes the total number of tests needed.

5.4 Parameter Definition

[Dor43, 3]

5.5 Non-Hierarchical pooling algorithms

Non-hierarchical algorithms involve testing over stages. The difference is that in this approach, individuals may be tested more than once per stage through overlapping pools [LTM+21, 3]

5.6 Optimal poolsize

There are a few methods to determine the optimal poolsize. The most that I have found during literature review did not consider cost in the retesting stage, meaning that the poolsize could potentionally be N, therefore you get 2 total transactions to do the process, one for ingest and one for the repairing. But then you would have to update your whole archive each time an object is found corrupted, since the pool securing the object is the whole archive. In my experiment, I have to find a way to add a cost to the optimal poolsize to prevent too large pool sizes. The optimal pool size to minimize write transactions on the blockchain is N, because if we can secure every object in the archive with only one write transaction. That would be very inpracticable in a real environment, because you would have to compute a new root hash on each update for each pool. Therefore shooting up the actions with every update for N actions.

CHAPTER 6

Fixity Storage

6.1 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. Fixity Information includes special encoding and error detection schemes that are specific to instances of Content Objects. Fixity Information does not include the integrity preserving mechanisms provided by the OAIS underlying services, error protection supplied by the media and device drivers used by Archival Storage. The Fixity Information may specify minimum quality of service requirements for these mechanisms [fSDS12, 4-30]. With fixity information you are able to answer questions regarding the authenticity and integrity of an object. *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. This supports assertions about the authenticity and trustworthiness of digital objects [KK$^+$17, 3]. 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 [KK+17, 38].

6.1.1 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, its 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 isnt provided as part of the transfer, you should create fixity information once you have received the materials [KKG+14, 4].

6.1.2 Fixity Checks

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 [KKG+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 [KKG+14, 2]. 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 [KK+17, 5]. For example, fixity checks may occur at different times depending on the institution's environment: during initial deposit only; during any file transmission; during scheduled backup routines; or periodically at specified times or when manually triggered [KK+17, 7]. *Throughput:* 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 ctivities can adversely affect other important functions like delivery of the content to users.

6.1.3 Fixity Instruments

In Table 6.1 For a given fixity instrument, 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's 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 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 [KKG+14, 7].

6.1.4 Storage Medium

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 running fixity information 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 [KKG+14, 7]. *Ethereum Blockchain:* In this thesis I haven chosen the blockchain to be the storage-medium for the fixity information. Its immutable state and availability in addition to its young usecase in digital archives have influenced my decision.

6.2 Interface

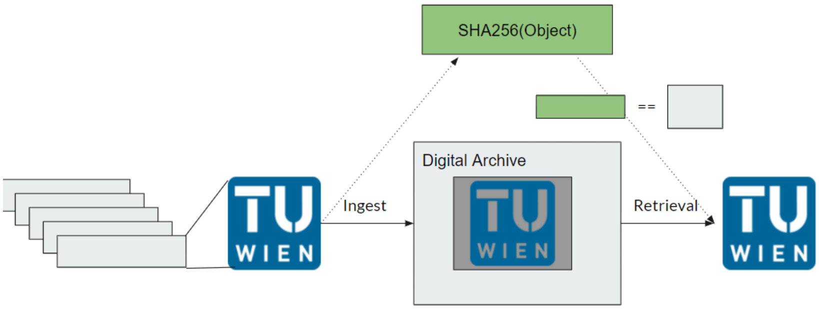
A fixity storage must persist any kind of fixity information, in my thesis SHA256 values, for longterm and guarantee that the persisted content is unaltered until retrieval of the content. I have chosen the Ethereum network as a medium for persisting file fixity information, see Section 6.1.4 for my reasoning. In Figure 6.1 the lifecycle of fixity information is presented, where at some point in time the information is ingested into 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.

6.3 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 6.1

```
    \label{lst:fixity-storage}
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.9.0;

contract FixityStorage {
  mapping(uint32=>bytes32) pools;
  address creator;

  constructor()public{
    creator = msg.sender;
  }

  function getPoolHash(uint32 poolId) public view returns(bytes32) {
      return pools[poolId];
  }

  function setPoolHash(uint32 poolId, bytes32 poolHash) public {
    require(msg.sender==creator);
    pools[poolId]=poolHash;
  }
```

Figure 6.1: Example of a fixity storage



```
}
```

Listing 6.1: MVP source code of the fixity storage deployed on the Ropsten test network https://ropsten.etherscan.io/address/0x18648B486Bd6B771DB957590E988A2464F22BfCd TODODODODO

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 fxiity 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 which fires a REVERT 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 creat and alter the information stored on the blockchain.

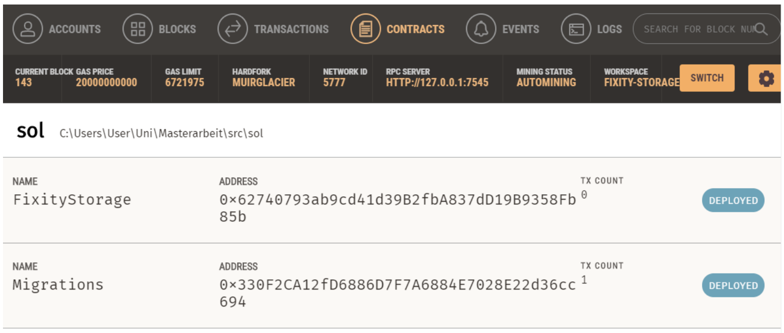
6.4 Deployment

I utilized trufflesuite[1] to run my deployment of the smart contract. I decided to use truffle to develop, test and deploy the smart contract for the fixity-storage. Truffle is a development environment, testing framework and asset pipeline for blockchains using the Ethereum Virtual Machine. Truffle 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 steps I used the user-interface Ganache[2] to get a better feeling for the ethereum blockchain, see Figure 6.2. It allows you to click through your smart contract and look at the state variables or functions to validate that your smart contract was 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 used 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 was resistent and showed no weakness when persisting 10000 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 someETHtoken in your account to pay the miner to integrate your smart contract in a block. Truffle offers some utility regarding automated deployment, which are migrations. Migrations are JavaScript

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

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

Figure 6.2: Ganache, an interactive user interface for the ethereum blockchain.



CURRENT BLOCK 143
GAS PRICE 20000000000
GAS LIMIT 6721975
HARDFORK MUIRGLACIER
NETWORK ID 5777
RPC SERVER HTTP://127.0.0.1:7545
MINING STATUS AUTOMINING
WORKSPACE FIXITY-STORAGE

sol C:\Users\User\Uni\Masterarbeit\src\sol

NAME	ADDRESS	TX COUNT
FixityStorage	0×62740793ab9cd41d39B2fbA837dD19B9358Fb85b	0
Migrations	0×330F2CA12fD6886D7F7A6884E7028E22d36cc694	1

files, which are responsible for staging the deployment tasks and running deployment scripts. I used the migrations feature in order to interact with various Ethereum networks, in my case the Ropsten test network and a local test environment. The migrations feature requires to have an smart contract deployed on the blockchain, which causes additional cost (191943 gas) to the fixity storage. The deployment cost of the smart contract can be calculated as follows:

$$C = G_{transaction} + G_{txcreate} + Contract_{bytesize} * G_{codedeposit} + G_{txdatanonzero} * Tx_{bytesize} \tag{6.1}$$

where $G_{transaction}$ is the base cost for a transaction; $G_{txcreate}$ is the operation used to create a smart contract; $G_{codedeposit}$ is the gas cost for each byte of the compiled bytecode of the smart contract; $G_{txdatanonzero}$ is 16 gas foreach byte in the compiled bytecode of a transaction; $Contract_{bytesize}$ is the size of the compiled bytecode of the contract; and $Tx_{bytesize}$ is the size of the compiled bytecode of the transaction, see Section 4.7 for the exact gas amount foreach transaction. The amount of gas consumed by the deployment of the decentralized fixity storage is 164779 gas. The location of the fixity storage on the blockchain is *0x18648B486Bd6B771DB957590E988A2464F22BfCd TODODODO*, where additional infos or the code can be read.

6.5 Authorized Access

In solidity one can require a certrain address to acess a function, the keyword requires can be used so that a transaction from a unknown address can be reverted and only the owner of the creator address can update the mapped SHA256 values in the smart contract. An interestng topic is also how the private key in the archive may be managed, which is not part of this thesis but something like a multisignature wallet may be used to split the responsibility of the owner address in the contract. Since the entity which controls the private key of the smart contract is able to perform update operations which can lead to unwanted actions. In the worst case, an unwanted action may be ignored, since the older value is not deleted on the blockchain. Therefore the key usermanagement of the smart contract can be used as a next steps in further research. For this thesis

I assume that the private key is well managed and each transaction coming from the master address is legit.

Table 6.1: Various Fixity Instruments [KK+17, 6]

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.

CHAPTER 7

Experiment

7.1 Setup

The experiment written in Python in form of a Jupyter Notebook. At first the function for providing the optimal poolsize was implemented, the exact implementation can range froma bernoulli experiment to another form. When the optimal pool sie is found, which is the econd research question I create 10000 objects and assign them to a certain pool and assign them a corruption rate ranging from 0.01 to 0.2. The corruption rate is the base for the bernoulli experiment. After object creation, the objects are ingested into the archive, durng this process each pool is persisted on the blockchain. This happens in an interaction between a python client and a local installation of the ethereum blockchain. Ganache is used for local development on the Ethereum side and pythons Web3 is used as a client. While ingesting and uploading onto the blockchain i will monitor the amount of gas used and the exact number of write transactions. The more eact method of monitoring the operation cost is by adding up the gas cost, since gas can be transformed nto ETH. The first measurement will be theoretical and the second empirical where i monitor the amount of ETH available for the experiment account. e.g. if i start the experiment with 100 ETH and at the end i have 90 ETH left the operation cost was 10 ETH. If the experiment local is successfulll I willl deploy the smart contract on the ropsten testnet and add time measurement to the porcess since the blockchain is now real and distributed on the network. I expect the time needed to ingest and upload on the blockchain to increase the cost should stay the same. When the objects are ingested into thedummy archive and the pools are uploaded onto the blockchain i will retrieve random samples from the original set of objects and check if they are corrupt, maybe i will assign corruption at random to this random sample and then I have to implement a repair function, this ffunction also affect the operation cost, since when a corrupted file is found the whole pool has to be re-computed resulting in poolsize + 1 local transactions and one blockchain write transaction. In the retrieval process i will monitor the time and nullify the cost since read operations on the blockchain are free of charge, nonetheless i will count the number of transactions needed for the process. After retrieval and repairing the pools i will look into the number of transactions needed and the time needed for the whole process. The result will be one row af a table and the experiment can be redone with different poolsies or corruption rates to get a nice table with descriptive statistics. At last I will split the objects and metadata and adapt the corruptions rates and redo the whole experiment and compare the results.

7.2 Object Identity

when and where should the pool id be stored, there are possibilities that the pool id is assigned with pool creation but that would mean that i have to update the object which is supposed to preserve, should the pool id be in the metadata of the object= if no, where should we store the link between the oject and its respective pool. Its easy to keep the links in the local jupyter notebook but in a industrial environment keeiping those links is rather hard. For now i keep the transaction hash with the pool object

7.3 Object

An object in this experiment is simply a SHA256 value, since the preservation process does not care if a picture, text or video is secured by the hash. An object also holds the reference to its pool, where in a real case the poolId is stored in the object's metadata. For the sake of the experiment, the initial bulk is numbered from 1 to N, where the SHA256 value is then assigned to the object. An object also has a float value which indicates the chance of it of being corrupted, this value is used in the bernoulli experiment to determine the optimal pool size based on the corruption rate of all objects. The final member variable of the object is a boolean flag which indicates wheter an object is corrupted or not.

7.4 Pool

A pool in this experiment is a collection of objects with size k. The root hash of a pool is a hash-list of every object in the pool.

7.5 Archive Mock

For the sake of the experiment I will mock the functions of a digital archive in python with the following relevant functions implemented: (1) $bulk_ingest$ (2) retrieve (3) $get_objects_by_poolId$ (4) repair. (1) Is for ingesting a bulk of objects with their respective metadata, it is expected that the poolId is already set for the object. The bulk itself does not know anything about the pool sizes or the amount of pools. The archive therefore is independent of the implementation of a pool and only need to know the poolId of an object. The poolId may be stored in the Preservation Description Information of an object [Lee10]. (2) Retrieve a single object from the archive. (3) The implementation of this function in a real digital archive may vary from my implementation, where I return every objects stored in the archive with matching poolId. This function is here to provide information regarding the original objects of a pool, this is necessary to rebuild the pool and recalculate the pool hash for someone who might want to check wheter an object in the archive got corrupted. (4) The function which handles the case when a corrupted pool was found. It is a costly function where one write transaction to the blockchain and additional data scrubbing has to be made. When a pool is found to be corrupted, each object in the pool is suspected to be corrupted too and therefore each object has to be replaced by a copy and reassembled in a pool, which hash is then again persisted on the blockchain. The mock also provides a function to simulate corruption, where each object in the archive has a chance to corrupt itself.

7.6 Program Flow

During development I have seen that the pool size should not be too high. When a corrupted pool is found, make sure to not double write the same pool to the blockchain You even got an advantage when you repair objects, since you have do scrub each object in the pool with a fresh copy.

7.7 Findings

in the cleaning process i cant count the reall number of corrupt objects because the number gets diluted by the fact that the local pools are false and even when we get a real object we could not recalculate the pool because the pool itself is corrupted therefore an uncorrupted object is seen here as corrupted since the pool cannot prove it anymore a positive sideeffect is if you find a corrupt elemen you have to scrub the whole pool, which means that the whole pool has is replaced with fresh copies There are massivele more repairing transactions

7.8 Dataset

I analysed the format-corpus [1] from Open Preserve Foundation [2] to get a better understanding of various file formats, mostly on how volatile they are. The latest commit for my analysis was *commit 4e4b9a34540f72612ba6eab2d28bccceb7a848ae*[3] on the 16th of February. 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

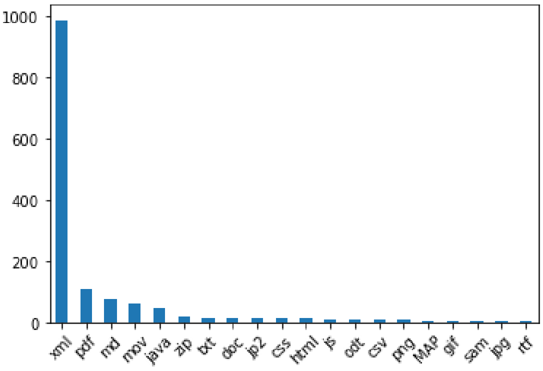
[1] https://github.com/openpreserve/format-corpus

[2] http://openpreservation.org/

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

corrupt links;were not available or did expect a tidious amount of time to download and re-strucutre them. Convenience; diversity of file extensions; and reputation of the organisation 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 7.1 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.

Figure 7.1: Distribution of file extensions in the format-corpus of Open Preserve Foundation. `https://github.com/openpreserve/format-corpus`



7.8.1 Transformations

To get a baseline estimate on how volatile certain extensions are due to updates and alterations, I analysed 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 occurance of a file 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. Latter 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 look into how often a certain file has changed without monitoring them on a system for a

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

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

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

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

certain time interval. Therefore I have decided to use the amount of file alterations in the git repository as a rough estimate. The estimated alteration rate p of a file extension is calculated with $positives/N$, see Table 7.1 I applied Eq.7.1, created by [REHHR20],

Table 7.1: Volatility of certain file extensions in the format-corpus dataset

extension	N	positives	p	k
xml	986	432	0.43	2
pdf	106	2	0.01	8
md	74	17	0.22	3
mov	61	0	0.00	61
java	47	39	0.82	2
zip	17	0	0.00	17
txt	14	2	0.14	4
doc	13	0	0.00	13
jp2	12	0	0.00	12
html	11	6	0.54	2
css	11	4	0.36	2
js	10	3	0.3	3

to each row in Table 7.1 to cacluate the optimal poolsize for each file extension.

$$k = 1.24 * (positives/N)^{-}0.466 \tag{7.1}$$

CHAPTER 8

Evaluation

The proposed fixity storage is evaluated on a local installation of the Ethereum blockchain and on the online Ropsten test network, where ETH used for transactions are free of charge. The two key parameters for evaluation are the operation cost and efficiency. *Operation cost* is calculated by multiplying the number of relevant cost transactions with the average gas cost of a writing transaction on the blockchain, see Eq. 8.1

$$C = J * Tx_c = \lceil N/k \rceil * (G_{transaction} + G_{txdatanonzero} * Tx_{bytesize}) \tag{8.1}$$

where J is the number of pools on ingest; Tx_c is the cost for a transaction in gas, see Section 4.7 for the gas cost of certain operations. I intent to present the operation cost in the form of gas, the unit that measures the amount of computation effort requried to execute specific operations, instead of the cost in USD or EUR. This is because, the amount of gas used during the experiment should stay 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* is measured by operations needed per object utilizing pooled testing compared with the individual testing strategy, where the efficiency of pooling strategy S is expressed by Equation 8.2

$$E(S) = N/T(S) \tag{8.2}$$

where, assuming the preservation process of n digital objects without pooling requires N operations and the preservation of the same objects using strategy S requires T(S) operations. In the case of indivudual testing the efficiency of E(N) is 1, whereas if strategy S requires two times less operations the efficiency E(S) is 2 [ŽLG21, 4].

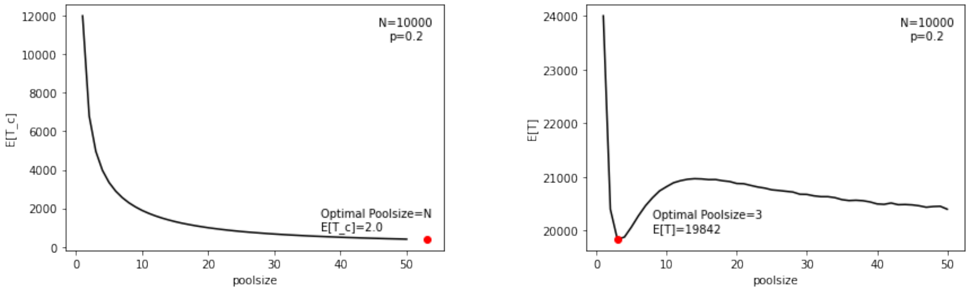
What is the optimal pool size based on the corruption rates of digital objects in the archive regarding cost and efficiency?

Corruption rate p represents the prevalence of corruption, e.g., when I assume that 2000 of 10000 objects will be corrupted during the preservation process, $p = 2000/10000 = 0.2$.

Expected 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 eq. 8.1a, because I only must write onto the blockchain during the ingest process where the root hashes of pools are persisted. After the first Iteration of the experiment, the number of corrupt objects per positive pools were included in the first draft of Equation 8.1, 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, then I can 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 poolsize calculated by Eq. 8.1 is N, see Figure 8.1a. A poolsize 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 poolsizes to avoid data scrubbing. Eq. 8.3 accounts in the number of times I have to perform data scrubbing in the archive, therefore the data scrubbing actions are included in the efficiency, since they have to be performed at retrieval in the archive.

$$T(S) = J + J_+ * k = \lceil N/k \rceil + ((1-(1-p)^k) * \lceil N/k \rceil) * k \tag{8.3}$$

where J is the number of pools and $J_+ * k$ is the number of digital objects in corrupted pools. Therefore the number of expected operations is the number of writing operations on ingest plus the number of data scrubbing operations on retrieval. By adding the amound of data scrubbing operations, the optimal pool size gets significantly lower, see Figure 8.1b The optimal poolsize k can be determined by finding the minimum of Eq. 8.3, which results in the highest efficiency in Eq. 8.2. In Table 8.1 it is shown that for ingest bulks with higher prevalence of corruption rate, smaller poolsizes are favorable. *To what extend can pooled object hashes increase the efficiency and reduce cost for a fixity information storage service on the Ethereum blockchain?* The efficiency of a



(a) Expected Cost

(b) Expected Throughput

Figure 8.1: Comparison of optimal poolsizes

Table 8.1: Expected transaction throughput with different prevalence of corruption rates

N	p	k	E[T]
10000	0.01	10	12052
10000	0.02	8	12929
10000	0.1	4	16799
10000	0.15	4	18475
10000	0.2	3	19842

pooling strategy were measured by savings of tests (in times) using a pooling strategy comparing with the individual testing of a target population. Assuming that the testing of a population of n individuals without pooling requires n tests and the testing of the same population using the strategy S requires T(S) tests, the efficiency of the strategy S can be mathematically expressed by

$$E(S) = N/T(S) \tag{8.4}$$

In the case of individual testing which requires to test all samples, E(S) = 1. If a strategy S requires two times less tests than the individual testing, then E(S) = 2. A larger efciency value means a better efciency of the strategy [ŽLG21, 4].

8.0.1 Evenly Distributed corruption rates

8.0.2 Context-Sensitive

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 neighbours, and is proven to be more effective than individual testing [DBK20, 4]. In this thesis, the idea of grouping similar files to increaes efficiency of the pooling strategy is implemented by grouping various file extensions, the results of in group efficiency are presented in table 8.2. In Figure 8.2, 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.

8.1 RQ 3 Given that metadata has a higher corruption rate, what effect has the split of metadata and objects on the operation cost?

Figure 8.2: Distribution file corruption by alteration in the format-corpus of Open Preserve Foundation. `https://github.com/openpreserve/format-corpus`

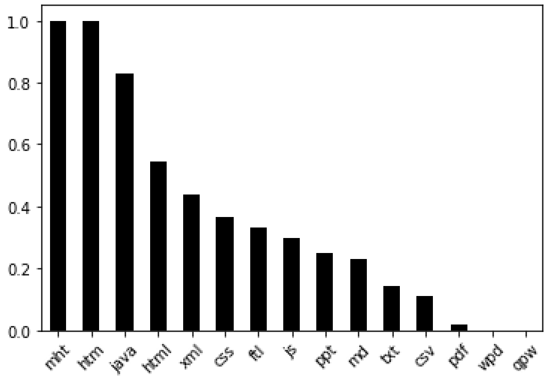


Table 8.2: In group efficiency of various file extensions

extension	N	positives	p	k	efficiency
xml	986	432	0.43	2	1.21
pdf	106	2	0.01	8	3.60
md	74	17	0.22	3	1.37
mov	61	0	0.00	61	61.00
java	47	39	0.82	2	1.21
zip	17	0	0.00	17	17.00
txt	14	2	0.14	4	1.33
doc	13	0	0.00	13	13.00
jp2	12	0	0.00	12	12.00
html	11	6	0.54	2	1.06
css	11	4	0.36	2	1.07
js	10	3	0.3	1.08	

CHAPTER 9

Discussion

CHAPTER 10

Conclusion

Bibliography

[B+13] Vitalik Buterin et al. Ethereum white paper. *GitHub repository*, 1:22–23, 2013.

[BCC+19] Tu Bui, Daniel Cooper, John Collomosse, Mark Bell, Alex Green, John Sheridan, Jez Higgins, Arindra Das, Jared Keller, Olivier Thereaux, et al. Archangel: Tamper-proofing video archives using temporal content hashes on the blockchain. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[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.

[fSDS12] Consulative Committee for Space Data Systems. *Reference Model for an Open Archival Information System (OAIS)*. 2012.

[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.

[HTBM17] Peijie Hou, Joshua M Tebbs, Christopher R Bilder, and Christopher S McMahan. Hierarchical group testing for multiple infections. *Biometrics*, 73(2):656–665, 2017.

[KK$^+$17] Kim, Katherine, et al. Fixity survey report: An ndsa report. 2017.

[KKG$^+$14] Kim, Katherine, Wayne Graham, Digital S Alliance, Aliya Reich, and Carol Kussmann. What is Fixity, and When Should I be Checking It. 2014.

[KORD10] Matthew Kirschenbaum, Richard Ovenden, Gabriela Redwine, and Rachel Donahue. Digital forensics and born-digital content in cultural heritage collections. 2010.

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

[LTM$^+$21] Nefeli Lagopati, Panagiota Tsioli, Ioanna Mourkioti, Aikaterini Polyzou, Angelos Papaspyropoulos, Alexandros Zafiropoulos, Konstantinos Evangelou, George Sourvinos, and Vassilis G Gorgoulis. Sample pooling strategies for sars-cov-2 detection. *Journal of virological methods*, 289:114044, 2021.

[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.

[Woo] Gavin Wood. Eethereum: A secure decentralised generalised transaction ledger berlin version.

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

[ZCL+20] Quanyu Zhao, Siyi Chen, Zheli Liu, Thar Baker, and Yuan Zhang. Blockchain-based privacy-preserving remote data integrity checking scheme for IoT information systems. *Information Processing & Management*, 57(6):102355, 2020.

[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.

Enter your text here.