PhD-FSTM-2020-41
The Faculty of Sciences, Technology and Medicine

Faculty of Engineering and Informatics

# DISSERTATION

Defence held on 16/07/2020 in Luxembourg

to obtain the degree of

## DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

## EN INFORMATIQUE

## AND

## DOCTOR OF PHILOSOPHY

## IN COMPUTER SCIENCE

by

Robert E. NORVILL
Born on 14 July 1989 in Leeds (United Kingdom)

# BLOCKCHAIN TECHNOLOGY FOR DATA SHARING IN THE BANKING SECTOR

## Dissertation defence committee

Dr. Radu State, dissertation supervisor
*Associate Professor, Université du Luxembourg, Luxembourg*

Dr. Irfan-Ullah Awan, co-supervisor
*Professor, University of Bradford, United Kingdom*

Dr.-Ing. Holger Voos, Chairman
*Professor, Université du Luxembourg, Luxembourg*

Dr Jean Hilger, Vice Chairman
*Banque et Caisse d'Épargne de l'État, Luxembourg*

Dr. Daniel Neagu
*Professor, University of Bradford, United Kingdom*

Dr. George Ghinea
*Professor, Brunel University London, United Kingdom*

# Blockchain Technology for Data Sharing in the Banking Sector

By

Robert Edward Norvill

A thesis submitted to
the University of Luxembourg
for the degree of
DOCTOR OF PHILOSOPHY


SErvices and Data mANagement (SEDAN)
Interdisciplinary Centre for Security, Reliability and Trust (SnT)
University of Luxembourg
May 2020

## ABSTRACT

Know Your Customer compliance costs have never been higher for banks in Europe. This thesis looks at the application of blockchain technology to reduce Know Your Customer compliance costs. The work within aims to utilise the strengths of blockchain technology in order to reduce the costs of compliance for banks. This is done through collaboration with industry partners, resulting in a system designed to meet banks' needs. The contributions of this work are: 1) A system which enables data sharing between banks, enabling 2) reduction of costs by at least 45%, and 3) reducing or eliminating over reliance on third parties, 4) an exploration of how to price data within the system is made in order to help banks further reduce their costs, 5) reduction of chain size by reducing the size of contract creation transactions in Ethereum by 90% for standard users, lastly, 6) to better understand the functionality and purpose of smart contracts. The system is the first of its kind to remove the requirement of third party storage solutions, and is the first to explore pricing aspects in detail.

**DEDICATION**

To Bob.

## ACKNOWLEDGMENTS

I would like to thank the people who have made this project possible, more enjoyable, or both:

My supervisor at The University of Luxembourg, Prof. Radu State for his ever-relaxed advice, chats about European history, and propensity for saying "OK, let's do it".

Our partners at ABBL for their continuous input into, and facilitation of, the project, especially: Marc Hemmerling, Jean Hilger, and Andrey Martovoy.

My supervisor at The University of Bradford, Prof. Irfan Awan, for his faith in me and always ensuring everything was kept on track.

Prof. Dan Neagu and Dr. Paul Trundle for the all opportunities, and advice they have given me over the years.

Dr. Andrea Cullen for her guidance, during my PhD.

Oliver Andryszewski for his proof reading efforts.

My SnT colleagues who have always been willing to help with the formation of ideas, and the consumption of coffee, during working hours. A special thanks to those who have been equally willing to aid in the consumption of beer afterwards.

My family, especially my mum and dad for all their care, support, encouragement, patience and advice over the years.

Last but not least, my wife, Claudia Parera, for always supporting me, and being far wiser, funnier, and more insightful than I could have hoped for.

# Table of Contents

# List of Figures

# List of Tables

# Chapter One

# Introduction

"Where there's muck, there's brass."

---

Yorkshire Proverb

## 1.1 What is Blockchain?

The work detailed in this dissertation makes use of blockchain technology. It is apt, therefore, to begin by detailing what blockchain technology is, and what it is not. Blockchains are decentralised systems, the primary function of which is immutably recording data in scenarios where participants do not fully trust one another. The properties of decentralisation and immutability can be either a strength, allowing for otherwise infeasible interactions to take place, or a weakness. What dictates whether blockchain's inherent characteristics are a boon or not is application. One needs only to apply blockchain technology to the right problem, in the right way.

Blockchains require multiple participants to function correctly. If there is only one actor then a conventional database solution should be sought. In the case of multiple participants, each one runs the requisite blockchain software and observes and records the blocks of transactions which are to be stored as part of the chain. Whether a transaction, and by extension a block, is legitimate or not is agreed upon by participants through a consensus

mechanism. A transaction is a message communicated to the blockchain system which either transfers valid in the form of cryptocurrency or activates a smart contract. Consensus is an important component of blockchain which gives the property of decentralisation, and therefore immutability. Immutability, in turn gives non-repudiation, which is an extremely useful property of blockchain, and one which is important for a lot of use cases, including ours.

Different types of consensus are suited to different types of chain and environment, this is discussed in more detail in section 2.1. Consensus is formed on each block that is proposed to be added to the chain. Each block contains a number of transactions, For a block to be considered valid all the transactions in it must also be valid, as well as some other meta data. Upon receipt of a block participants check its validity of it before adding it to their copy of the chain; this process varies depending on the consensus method being used. The oldest method of achieving consensus is Proof of Work (PoW), which is common on public blockchains. It exists to avoid a blockchain system being controlled by one entity, or a number of colluding entities. If a straight forward voting system is used, then an individual can simple create more blockchain identities to outnumber the other voters, and in doing so control the vote by taking 51% or more of the total votes. PoW mitigates this by making use of a mathematical puzzle; which is computationally difficult to find a solution for. Anyone wishing to add a valid block to the chain must first find a solution to the puzzle, in doing so they incur costs in the form of hardware and electricity usage. Miners are rewarded for their work producing blocks in the form of the cryptocurrency native to the chain in question, more details on this can be found in 2.1. The idea is that it becomes prohibitively expensive for any one entity to amass enough power to control 51% or more of the power. If colluding entities did manage to alter a block, or amassed enough computing power to do so, it is likely to be noticed by other users, this heavily devalues the worth of the cryptocurrency in question, which also strongly incentivises honest behaviour. Real life examples have shown this to be true, in 2014 miners voluntarily left a pool that was approaching a 51% of the

mining power over fear of it devaluing the cryptocurrency [43]. Crucially, while the puzzle is difficult to solve, it is easy to check the validity of a solution once it has been found.

However, PoW is not suited to all types of blockchain. It is very good at protecting public blockchains where anyone who wishes to can join, and participants are neither known nor trusted. The main drawbacks of PoW are the cost incurred in terms of hardware required to solve the puzzle, and the electricity this consumes. The difficulty of the puzzle necessitates a waiting time for a puzzle solution for each block, which means a waiting time on transactions. Lastly, there are environmental concerns that stem from such a vast amount of electricity being used, with it being estimated that Bitcoin's annual electrical energy consumption outstrips that of The Czech Republic [20].

When using a private chain it is possible to dispense with PoW and use less costly voting systems instead, such as Proof of Authority (PoA). If there is 100% trust between all participants then a blockchain may not be the best fit, as it would be possible to have a 100% trusted entity host a database containing transactions and account balances, etc. On the other hand, if one is dealing with marketplace rivals who wish to collaborate in order to mutually reduce costs then you have a semi-trusted environment and a PoA backed system can be used. As opposed to public chains, private permissioned chains are ones for which access is controlled and participation can be curated, all participants are known, and who is allowed to vote and view the system can be controlled. This removes the need for the expense based protection provided by PoW. Under PoA a block is only considered valid if the transactions in it are all valid and it has been signed by an account which is recognised as having the right to vote. Hence, any participant proposing a block needs to prove their authority to do so. This is much faster and cheaper than PoW as there are no expensive puzzles to solve.

Participants are incentivised to behave honestly as they wish to interact on the private permissioned chain in order to receive the benefits that come from doing so, for this project the benefit is reduced costs. Known identities and the desire to continue to remain within

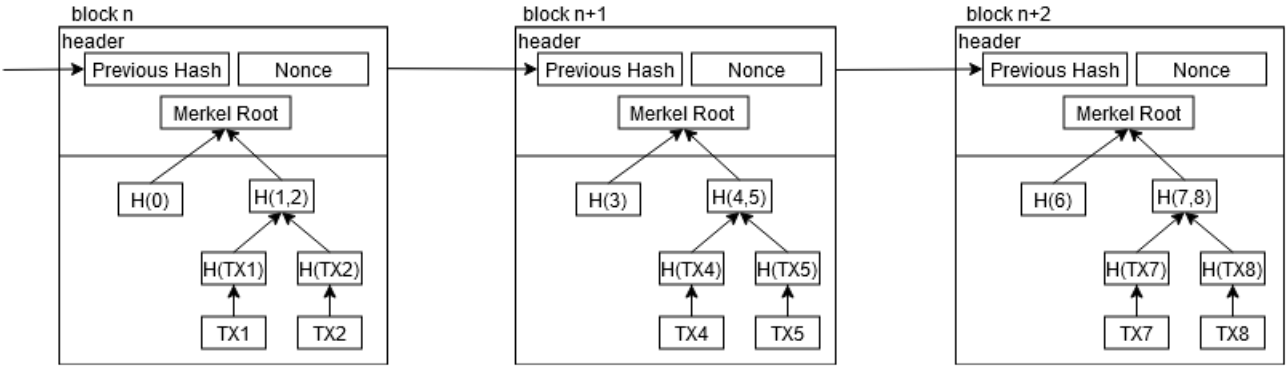


**Figure 1.1** Ethereum block links, block bodies and headers

the system provide assurance of honest behaviour. No new identities can flood the system, and any dishonest voting will be quickly identified, along with the offending party. Dishonest/invalid transactions, or dishonest changes to previous blocks, would be witnessed by the other participants. This can result in off-chain consequences beyond removal from the system, the exact nature of which depends upon what was contractually agreed amongst participants.

In summary, PoA voting has the advantage of being much cheaper than PoW. It does not require massive power consumption to solve puzzles. Nor do participants have to wait while a solution to the puzzle is sought, making majority voting much faster. Types of consensus and the roles of blockchain actors are discussed in more detail in section 2.1.

Each block in the chain must contain the hash of the previous block. A block's hash is generated over all of a block's contents, including the hash of the previous block. In this way, each block is dependent on the previous block.Moreover, each transaction in the block is hashed, and the resulting hashes are hashed together until there is only one hash, this is known as a Merkel tree, with the final hash being considered the root hash, or Merkel root. Each block includes a nonce (unique random number) which is a valid solution for the PoW puzzle for that block. A change to any of the contents of a given block would require all subsequent blocks to be changed as well due to the hash of each block changing in turn. As an example, block contents and use of hashes in the Ethereum system can be seen in

Figure 1.1.

When participants are checking whether blocks they receive are valid, they will calculate the hash over the block's contents. As every block is added by consensus, and participants are incentivised to behave honestly with their voting, it becomes practically impossible for any one entity to change a block. No rational actor would agree to remove valid blocks in order to make changes. They stand to gain nothing from doing so and, could incur new costs or be removed from the system. As long as the consensus remains distributed, whether through PoW or PoA, blocks cannot be changed.

The difficulty of changing blocks increases with the number of subsequent appended blocks, as each block requires the majority consensus of the system and blocks are linked to one another. This is the concept of weight; the deeper a block is (the more subsequent blocks have been appended) the less likely it becomes that the block in question will change. In figure 1.1 block $n$ has a greater weight than blocks $n+1$ and $n+2$. The concept of weight exists in both public and private chains. In public chains weight is backed by the cost of block creation, however in public chains it relies on the fact that participants are known, and can be banned from the system.

Given the ever increasing difficulty of changing or removing a block, it is common to say that blockchains are immutable. Any break in the chain would render all blocks after the break essentially void in terms of the on-chain truth. It is this inability to remove anything from the chain that allows participants to interact or transact with one another without the need to either fully trust each other, or, when smart contracts are utilised, to trust each other at all. Anything added to the chain will remain there, visible to all participants, in chronological order. Moreover, all transactions are observed and agreed upon before they are added. This gives the property of non-repudiation, meaning that no actor can disassociate themselves from their action, nor can they retroactively change the record of that action.

Blockchains often replace a trusted middle man, the ability to transact in this way has been at the core of blockchain since its inception and is why decentralised systems are often

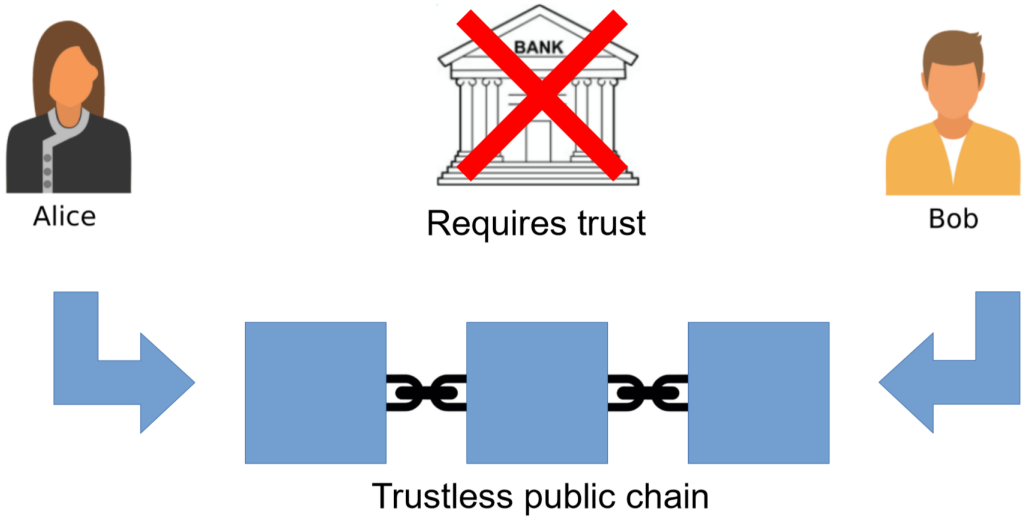


**Figure 1.2** Trustless blockchain transactions remove the need for a trusted entity

referred to as trustless; there is no one in the middle of the trade/interaction that requires the trust of those wishing to trade. For the first generation of cryptocurrencies like Bitcoin, this primarily meant removing the trusted entity (the bank) from financial transactions between entities, as shown in figure (figure 1.2). However, for second generation systems like Ethereum, the scope of what could be made trustless vastly increased.

## 1.2 The burden of compliance

### 1.2.1 Know Your Customer

Know Your Customer (KYC) refers to the legal requirements that certain institutions and businesses must comply with in order to be considered to have done their due diligence with regard to knowing who they are doing business with. Although various types of business are bound by KYC regulations, for the purposes of this thesis the focus is the banking sector. Within the European Union (EU) there has never been a greater burden of compliance placed on the banks than there is at present. Increased regulation means increased costs for banks as they have to do more to ensure they remain in compliance.

In general, there are several things that banks must do in order to be compliant:

1. Collect documents from end-clients
2. Verify the validity of the documents
3. Store copies of the documents
4. Update documents at certain intervals or before expiry

Exactly which documents are required, what the verification process is, and when they must be updated depends on the service desired by the customer, the type of customer, and the bank's own internal rules. It is possible for banks to have different documentation requirements for the same service. Customers can be split into natural persons and legal persons. Within these two categories there are several different types that can affect the requirements listed above, these types can be seen in table 1.1. They are given from the point of view of a bank within the EU, and written in ascending order in terms of complexity of KYC procedures. A Politically Exposed Person (PEP) is anyone who holds a significant public, political position, or plays a significant role in a government's operation (senior civil servants for example). These people are considered to be higher risk in terms of likelihood of involvement in corruption and/or bribery due to their position. As such, the burden of KYC is higher, as checks increase and Anti Money Laundering (AML) laws play a larger role than they would for an average citizen. Terrorism is an exception as no service whatsoever is given to any person connected to terrorism, or the funding thereof.

Typically, there is little or no cooperation between banks when it comes to KYC as banks are market place rivals, and each is individually responsible for its own KYC compliance. This leads to a situation whereby there is massive repetition in KYC work, both for banks and their customers. As a simple example, if a customer has an account with one bank and wishes to open an account with another they will have to repeat the process of presenting documentation to the bank in order to prove their identity. In turn, the bank will have to

| Natural Persons | Legal Persons |
|---|---|
| National citizen | Sole trader |
| EU/EEC citizen | Small company |
| Third country citizen | Medium company |
| Politically Exposed Person | Corporatation |
| Terrorist | - |

**Table 1.1** Different types of bank customer

repeat the processes already carried out by the customer's existing bank of verifying, copying, and storing the documents. Moreover, the processes the banks have in place to verify and copy documents are often manual, making them slow and costly. When the customer is a corporate entity the process is more burdensome for both sides, with the amount and complexity of the required documentation increasing.

The repetition of KYC processes is the main focus of this project when it comes to reducing the associated costs. The aim is to provide a system whereby once a document has been collected by any one participating bank, it can be received by the other participating banks, provided that the customer gives explicit permission for the document(s) to be shared with a bank. Given that so many of the KYC processes are still manual, there is a secondary focus on automation in order to further reduce costs.

### 1.2.2 General Data Protection Regulation

The EU's General Data Protection Regulation (GDPR) is of key importance for any company which handles personal data and operates within the EU. As blockchain technology is generally considered to be immutable, extra care must be taken as to what and where data is stored in order to ensure alignment and (where possible) compliance.

The GDPR [101] defines a number of actors when it comes to the ownership and handling of personal data, these are summarised in table 1.2. The Data Owner (DO) is the person

| Actor | Definition |
|---|---|
| Data Owner | Person who decides which entities have access to their data |
| Data Controller | Any entity which holds and processes personal data |
| Data Processor | Any entity which processes data on behalf of a Data Controller |

**Table 1.2** Definitions of actors according to The GDPR

who decides which entities have access to their personal data. A Data Controller (DC) is any entity which holds and processes personal data, in the case of this project, this is primarily the banks. Lastly, a Data Processor (DP) is any entity which processes data on behalf of a DC. For the use case at hand, this might be seen in two ways; when a bank chooses to outsource part of its KYC processes to a third party, or when banks share not only data, but also the results of KYC processes and checks. In this scenario it is possible that a bank would be considered a DP for another bank.

As the DO decides which entities can have access to their personal data, it is essential that banks gain the explicit permission to hold the necessary data from each of their customers. Moreover, under the GDPR the DO has the right to request the deletion of their data. If a request for deletion is made the DC must comply, proving there is no overriding legal reason not to. For example, in the EU AML legislation [100] takes precedence over the GDPR. Therefore if a bank is required under AML legislation to retain a client's personal data for a certain period of time then a GDPR deletion request would not be fulfilled.

With regard to blockchain, care must be taken to ensure GDPR compliance. Due to the need to delete personal data at the request of the customer, no personal data can be stored on the blockchain. However, it is sometimes necessary to store identifiers for customers on the blockchain, as without them it would not be possible to identify which customer had

given or revoked permission for which bank. In such a scenario it is appropriate to look at methods of anonymisation. If any on-chain data could be legally considered to be personal data then cryptographic methods must be used to ensure that the on-chain data can be fully anonymised in the event that a bank is legally required to delete a client's personal data.

When the GDPR was originally published, it was not clear what would be required for blockchains to be compliant. The requirement that personal data can be deleted did not fit well with blockchain's immutability considering hashes of personal data were also likely to be considered personal data. In July 2019 the EU's Scientific Foresight Unit published a study which asked the question "Can distributed ledgers be squared with European data protection law?" [33]. The study finds that blockchain can not be said, as a whole, to be either compliant or incompliant with the GDPR, but that compliance can be assessed and attained on a case-by-case basis. Moreover, the study stressed the need for clearer guidance at the EU level for the use of blockchains and their compliance.

Some clarity was provided in October of the same year via a document released by the Spanish Data Protection Agency and European Data Protection Supervisor [18]. It states that hashes can be made fully anonymous providing they are used in the right way, and make use of salts. Salts are single use, secret, random strings which are used as an input for hash generation along with the identifiable information (name, passport, address, etc). According to the document, if the data is erased such that even the DC cannot recreate the hash, and thereby identify the DO by it, then it has been fully anonymised. In other words, if both the data and the secret salt are deleted then neither the DC, nor anyone else, can recreate the hash, and it can now be considered anonymous data. This insight makes it clear what blockchains must do in order to be considered compliant when storing hashes on-chain. This is highly relevant for the work in this thesis which makes use of hashes of personal data in order to ensure the validity of data within the system.

## 1.3 Research Questions

- Questions should cover contributions
- Question should be linked to chapter conclusions in your research objectives
- Questions should be linked/mirrored in final Conclusions to show how I met the original aims
- Chain size
- Solution
- Pricing
- Know your smart contracts

## 1.4 Methodology

- Needs to include a planned review of state-of-the-art work
- Identification of knowledge and technology gaps
- Followed by research on blockchain performance etc.
- Case studies to validate theoretical contribution to knowledge in an order that reflects thesis structure

## 1.5 Contributions

The main contribution of this thesis is a blockchain-based data sharing system for KYC data designed specifically for the banking sector. The application of requirements engineering has resulted in a system built to meet specific industry requirements, with security/data

protection by design. The overall purpose of the system is the reduction of costs associated with KYC.

The system encapsulates several achievements:

- Reduction of over reliance on third parties.
- The creation of a new form of cooperation between banks.
- Compliance.
- A disruptive business model in the form of a new marketplace.
- Data pricing which allows the banks to generate revenue in an area where it was previously not possible to do so.

Over reliance can be extremely costly for banks, reducing or eliminating it helps them to reduce costs. This thesis provides a novel blockchain base KYC data sharing solution, which is detailed in **??**. The solution is augmented through the creation of a marketplace in which banks can trade data allows them to reduce or recover the costs they incur through KYC when coupled with the ability to price data properly. It is necessary to carefully consider data under our system as it facilitates a unique marketplace within which nothing is traded without the end-client's permission, the buyers and sellers are curated, and are of potentially extremely different sizes. By way of contribution, an exploration of the factors which affect pricing, and different ways of pricing is detailed in chapter 6. Of course the system and its marketplace cannot exist if they are not compliant with the relevant legislation; the system is built in line with current EU level guidance and law in order to ensure compliance.

In addition to the contributions detailed above, this thesis includes a method for the reduction of the size of Ethereum based blockchains in chapter 4. As chains generally only ever grow it is always worth looking to the future and considering methods to reduce the required storage. Lastly, in chapter 7 a method which financial institutions can use to better

understand the purpose and function of smart contracts which they interact with is detailed. Through doing so they can ensure their on-chain interactions are safe and compliant, even in the face of potential future legislation.

## 1.6 Thesis Structure

The rest of this thesis is organised as follows: chapter 2 expands on what blockchain is and what smart contracts are. It gives more detail on the problems surrounding KYC. chapter 3 provides a literature review and highlights what is missing in the context of KYC for the banking sector. Chapter 4 details a method to enhance the performance of blockchain by proposing a method to reduce the size of Ethereum based blockchains. This is something that could become more relevant to banks over time as they size of the chain they use to interact increases. Chapter 5 discusses the main contribution of this thesis, the blockchain-based system designed for banks to use to share KYC data in compliance with applicable laws. Notable achievements of the system include cost reduction, and removal of over reliance on third parties. This chapter also introduces the concept of pricing data in order to further reduce KYC costs. Chapter 6 goes into the topic of pricing in depth and details different actors' requirements for pricing. Different methods of pricing data, and the impact they might have on the marketplace are discussed. Chapter 7 looks to the future, and anticipates, the possibility of legal requirements to Know Your Contract, as use of smart contracts increases. A method to help identify the purpose and functionality of smart contracts is detailed as a method to enable banks to better understand the technology they interact with. Finally, the conclusion is found in Chapter 8.

## 1.7 Supporting Publications

This section lists the publications which are relevant to each chapter.

- **Chapter 4:** Robert Norvill et al. "IPFS for reduction of chain size in Ethereum". In: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). IEEE. 2018, pp. 1121–1128.

- **Chapter 5:** Robert Norvill et al. "Blockchain for the Simplification and Automation of KYC Result Sharing". In: 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). IEEE. 2019, pp. 9–10.

- **Chapter 7:** Robert Norvill et al. "Standardising smart contracts: Automatically inferring Ethereum Request for Comment (ERC) standards". In: 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). IEEE. 2019, pp. 192–195.

- **Chapter 7:** Robert Norvill et al. "Automated labeling of unknown contracts in Ethereum". In: Computer Communication and Networks (ICCCN), 2017 26th International Conference on. IEEE. 2017, pp. 1–6.

- **Other:** R. Norvill, B. B. F. Pontiveros, R. State, and A. Cullen, "Visual emulation for ethereum's virtual machine," in NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium. IEEE, 2018, pp. 1–4.

# Chapter Two

# Background

> "As a general rule, the most successful man in life is the man who has the best information."
>
> Benjamin Disraeli

## 2.1 Types of Blockchain, Actors, and Consensus

This section takes a more in-depth look at some public and private chains, as well as consensus. An understanding of the different roles that participants can take, the functions these roles serve, and how they might suit different scenarios is necessary. When discussing blockchain and its uses it is important to understand the different types of blockchain that can be implemented and the different types of consensus which may suit these, an overview of this is given in section 1.1.

The biggest distinction to be made when looking at different types of blockchain is that between public and private. A public chain is completely open, anyone in the world can join in and run a node (blockchain software), providing they have an adequate internet connection, and a computer with enough storage space/speed. Famous examples of such systems include the Bitcoin and Ethereum main-chains. Such open, public chain systems are generally the ones on which cryptocurrencies exist, due to their completely trustless

nature. Generally speaking, because such systems do not rely on any trusted entity, or even a selected set of trusted entities to operate, everyone is able to trust in the system as a whole. This is known as a trustless environment.

In such an environment mechanisms need to be in place to ensure the honest behaviour of participants. For public chains this means building the system such that the rewards for honest behaviour are greater than the gains for dishonest behaviour. To put it another way, dishonest behaviour is too expensive to engage in.

### 2.1.1 Miners

A number of different roles are common on public chains. Miners are participants who attempt to solve the PoW puzzle in order to add new blocks to the chain. Miners collect transactions which they are aware of into blocks. The standard PoW puzzle involves varying a nonce in order to generate a hash with the contents of the block and the nonce as inputs. A nonce is a single use random number. The resulting hash must have a specified number of preceding zeros. The more leading zeros required, the harder the puzzle is to solve. Once a miner finds a solution they announce the hash, nonce, and block to the system at large, at which point other miners and standard users will check if the solution is valid by checking each transaction in the block and generating the hash for themselves. It should be noted that each block includes the hash of the previous block so an incoming block would be rejected by users if it did not match match with the current latest block. Miners carry out this process in order to be paid in block rewards.

Miners earn a block reward by successfully solving the PoW puzzle. A Block reward is a special transaction included in each block that is paid to the account of the miner that solves the PoW puzzle. Here is can be seen that miners are financially incentivised to produce valid blocks. The network at large has no reason to accept an invalid block, so a miner will only include valid transactions in their block and only announce valid solutions. To do otherwise

will not earn them money, and could result in them being black listed.

On a private chain, it is still necessary to have the role of miner, as they are still required for the proper functioning of the chain. However, on a private chain it is often possible to dispense with PoW. As participants are selected, and the system is not open to any actor anywhere in the world, there is a move from a zero-trust environment to a semi-trusted environment. The incentives for honest behaviour change somewhat, as do the threats that the chain needs to be protected against. In such a situation PoA may be used instead. Strictly speaking there is no mining under PoA and those generating blocks are often referred to as verifiers instead, this is discussed in section 2.1.4.

### 2.1.2 Standard Users and Archive Nodes

Standard users hold a full copy of the chain and check the validity of new blocks they become aware of and add them to their local copy of the chain. A standard user's interest in running a node is that they are able to easily fully validate transactions which they are interested in without relying on a third party. They have no interest in accepting invalid blocks from miners, as these blocks are no use for them.

In some systems, such as Ethereum, there is a lot of extra data that is not held by standard users, as it become prohibitively expensive to do so. For Ethereum this is previous chain states, with the global state being made up of the individual state of every account and contract at any given time. Instead, standard users store only the latest state in full and can request previous states from archive nodes, should they need to use them for validation. Hashes of states are stored as part of each block in Ethereum, so a standard user is able to check the validity of the states they receive.

Archive nodes (or full nodes) are most often run by companies whose business is based on blockchain and cryptocurrencies, or enthusiasts or researchers who run such nodes for their interest and work. These nodes hold all possible information about the system and its

previous states, including orphaned blocks, which are groups of blocks that were worked on but ultimately got overtaken by another set of blocks, which became considered the main-chain. These nodes require a lot of storage space to run, given the speed at which new blocks are produced; the write speed of storage solutions can also be an issue.

### 2.1.3 Light Clients

There is another type of participant, which is at the other end of the spectrum to an archive node: the light client. This type of client does not store whole blocks, only block headers. In Ethereum, upon learning about a new header a light client simply generates the block hash from the information contained in the header, and adds the header to their copy of the chain if it is valid. These nodes rely heavily on full or archive nodes to carry out full validation, and may download specific full blocks that they are interested in. Ultra-light clients also exist, they do not store any form of chain at all and simply rely on other nodes to check the validity of anything they need to know on a case by case basis. Although these types of nodes are not doing much in the way of helping to maintain the ecosystem of the public chains they are part of, they are often appropriate for low-power and mobile devices which have a need to interact with the chain.

### 2.1.4 Public, Private, And Consensus

What makes PoW necessary in public chains is the need to prevent an entity taking control of 51% or more of the votes, and thus controlling the entire system. If it were done simply as one account one vote, then the open nature of a public chains allows for potentially any entity to create a large number of accounts and control the vote. However, as the PoW puzzle is very expensive to buy hardware for, and expensive in terms of electricity consumption to solve, controlling the system would require an extremely large investment, infeasible for most. In addition, an entity that did amass enough computational power to control 51% of

the vote risks seriously devaluing the currency as its value lies largely in the fact that it is decentralised.

When the participants, and number thereof, are actively controlled, PoW is not strictly necessary. When it is possible to control who mines (produces blocks), the system can take advantage of faster, cheaper methods of achieving consensus. When using other methods of consensus, the miners are more accurately referred to as verifiers or voters, as there is no mining taking place. If who can send transactions and generate blocks is controlled, but the chain is publicly visible to all, then it is semi-private. If who can transact, verify and even see the chain is controlled, then the chain is private.

In Ethereum it is possible to run a private chain with a curated list of verifiers. The built in consensus method for doing so is called PoA. Under this system a list of accounts which are allowed to create blocks is defined in the genesis block (the first block in the chain). Blocks are only considered valid if they are signed by one of these accounts, in other words these accounts have the authority to produce blocks. By signing a block they are proving they have the authority to produce the block. This is much faster and cheaper than PoW as no specialist hardware is required and no time or electricity needs to be expended on puzzle solutions. However, verifiers also have nothing (financially speaking) at stake, so it is not a suitable system for a public chain. In a private chain scenario however PoA can work extremely well, if the participants are known and curated then expulsion from the chain and/or other off-chain consequences for misbehaviour is enough to keep verifiers honest. In order to add or remove a verifier, an address must be proposed by one of the existing verifiers. All verifiers can then vote on the proposed addition or removal, with 51% being required for the proposal to be acted upon.

Before moving on, it is worth looking at Ripple. A somewhat tangential system, Ripple makes unconventional, and even controversial, use of some of the properties of blockchains systems. Although Ripple plays host to a cryptocurrency (XRP), it is not a blockchain, rather it is an open source implementation of a protocol. Originally described in a whitepaper [87],

the updated version of the protocol is detailed in a second paper published in 2018 [12]. The system acts as a replicated state machine, with each node holding the current state. The state is defined as the current balances of all accounts on the network. Ripple makes uses of a distributed ledger (database), that stores the agreed upon account balances. Consensus is achieve by 80% agreement amongst trusted validators, almost all of which are under the control of Ripple, at the time of writing.

Validation is a simple voting process, as, under Ripple, validators are trusted, and so reliance on protective methods like PoW are not necessary. However, trust in validators goes a long way towards centralising the system, a potential weak point. Interestingly, validators are not paid for their work. Generation and distributed of the XRP cryptocurrency is entirely at the discretion of Ripple itself. Moreover, the majority of the trusted validators are controlled by Ripple at the time of writing. Given the above, Ripple blurs the line between centralised and decentralised, providing a publicly traded cryptocurrency which is backed by what is at least a semi-trusted system. However, it is worth noting that unlike the other big public chains, Ripple is focused on providing low latency and low fee transactions when exchanging currency or assets. Its confirmation time is around 4 seconds, rather than the recommended 6 blocks, around one hour, in Bitcoin [105]. Due to its focus on rapid settlement and low fees, it eschews smart contracts, and has no implementation of them.

## 2.2 The Advent of Smart Contracts

Bitcoin was the original blockchain, and as such it marked the start of the first generation of blockchain technology. Bitcoin's primary function was, and remains, to facilitate the transfer of funds between entities without the need for a trusted third party. Although it supports a very limited scripting language which allows for the execution of more complex transactions, it does not support smart contracts, which were the major contribution of Ethereum when it launched in 2015. With Hyperledger Fabric launching later the same year, an explosion

in the possible use cases for blockchain had begun.

Smart contracts are general purpose on-chain programs which run when a transaction is sent to them. Although it should be noted that they differ in some way to programs that are run by a single user. Most notably they inherit the blockchain's properties of consensus and immutability, meaning all previous states of every Smart contract is known and that, upon execution, participants all run the programs with the inputs given in the transaction and agree upon the execution and output. This means that the execution is trusted, as it is governed by consensus. It has the side effect that there can be no randomness in smart contract execution, as this would make participants unable to agree on what the outcome of execution was. It should be noted that the inputs given to the smart contract, every step of execution, and the outcome are all visible to those who can view the chain upon which the contract runs.

Smart contracts allow for operations that were once carried out by a third party to be moved to the blockchain, where trust in such an entity is no longer required. In some cases, this allows one entity to decouple itself from another which it would rather not pay or trust. In other scenarios it allows for new collaborations to form between entities which have an interest in cooperating (cost reduction, for example) but do not trust each other, as is the case with market place rivals. The choice between the use of a public and private chain is important when considering the use of smart contracts. The use of a public chain would allow any observer to trust the execution and outcome of a smart contract, yielding the most possible trust in the operation. However, it is possible that a smart contract is used for some process or interaction which is considered sensitive by the persons or businesses involved. In this case a private chain is better suited, where interested parties participate in the chain and form a consensus, without the world at large being able to observe.

Real world examples of public smart contracts providing services include the Ethereum Name Service (ENS). At the heart of the ENS is a smart contract which keeps a trusted record that acts as a DNS for Ethereum, recording who owns which name and allowing users

to have Ethereum specific, human readable addresses assigned to their Ethereum wallets addresses: "With ENS, you'll be able to send money to your friend at 'aardvark.eth' instead of '0x4cbe58c50480...'" [63].

A creative and highly profitable utilisation of Ethereum's smart contract technology has been CryptoKitties [52]. At the core of the system is a smart contract based on an ERC, which is a category of Ethereum Improvement Proposal (EIP) [34] that focuses on standardising smart contracts. In particular Cryptokitties utilises ERC-721, which defines a non-fungible token standard [35]. This is interesting because cryptocurrencies generally rely on the fungibility of tokens, meaning that every 1 ether (Ethereum's native cryptocurrency) is equal to any other 1 ether. Conversely, non-fungible tokens are not interchangeable, which opens up a different range of possibilities and allows for the creation of uniqueness. More information and some of the industry entities supporting ERC-721 can be found at the ERC's own website [1]. Cryptokitties defines tradable cats as ERC-721 tokens, which can be bought and sold between collectors. The creators also find a cryptograph way to allow users to 'breed' two digital cats, producing a new unique one. This has been a profitable on-chain game for the company behind it, with $12 million in sales being reported by the end of 2017 [108]. Some high level details about the system can be found in the white paper [94].

Ripple hosts a cryptocurrency without a blockchain. In keeping with this theme there is a system that allows for remote, distributed execution of code (smart contracts) without a blockchain, Codius [96]. Instead it acts as a system whereby hosts run virtual machines upon which arbitrary code can be run. If the parties wishing to run code trust the host they need only upload it to be run by them. If the host is not trusted execution can be duplicated across multiple hosts, the more hosts run the code, the more the result of execution can be trusted. The strengths of such a system are allowing users to write code in any language they please and the containers can execute arbitrary code. In addition, execution may be quicker than on a public chain where the entire network must run the smart contract code and agree upon it. The downside, however, is that it involves another layer of complexity:

additional software and payment systems for the user to deal with, as well as coding in interaction with blockchain systems if on-chain confirmation is required. Codius has been utilised by Coil, a business which designed a model whereby Internet content is paid for by streaming micropayments, rather than by advertising [95].

## 2.3 The Problem of Over Reliance

Many businesses and technological solutions are centralised. There are scenarios were centralisation is generally not considered a problem. For example, customers trust their bank to make payments on their behalf and provide an account into which their wage can be paid. Moreover, if a customer prefers the services offered by one bank over another they are free to switch between them.

This is not the case for a typical centralised solution for reduction of KYC costs. In such a scenario the bank outsources its KYC processes to a trusted third party, meaning they pay a the third party to collect, verify and store all their KYC related documentation. This posses a number of issues for the bank. Firstly, they are the ones legally responsible for KYC compliance, so if the third party were to make a mistake the bank would still bear the legal costs and consequences. Secondly, the bank must now rely on the third party for document collection, verification and storage, which means the third party is the sole provider of essential services to the bank, and, that the provider holds the bank's data on their behalf. Being so closely tied to a provider means they have become over reliant on them.

Over reliance on third parties is extremely costly for banks. A third party upon which the bank relies is able to increase the cost of the service they provide as the bank does not have a choice about using the third party. Such third parties include name list checking services, the banks must use these by law, and KYC outsourcing systems that store the bank's data for them. In the case of outsourcing the bank is exposed to a high risk of significantly increased

costs as disengaging the provider would potentially require a prohibitively large transfer of data from the provider bank to the bank. The bank must either have a new provider in place, to which the data can be sent, or prepare its infrastructure to handle all data storage and KYC procedures. Either way, changing providers for such a service is likely to be costly, time consuming and even threatens a disruption to the bank's day-to-day operations for things like on-boarding new customers.

As can be seen in section 3.3, current academic studies all rely on (at least partially) centralised storage solutions. The system detailed in this work comes with the ability to interface with banks' existing storage systems, enabling them to continue to store their own data, and provides a system that is minimally disruptive to the banks' IT infrastructure. As such, the system gives them options for the reduction of KYC costs which do not make them overly reliant on an particular entity that could expose them to higher costs in the future.

Reliance on checking services is potentially dramatically reduced if the banks trade the results of such checks between one another as the banks would be less reliant on such services. Such a large shift away from over reliance, coupled with the banks trading amongst themselves, makes the system a disruptive one in the area of FinTech (Financial Technology).

Figure 2.1 highlights how the fully outsourced model has the same amount of duplication as when the banks handle their KYC in-house. The customer must send the documents twice, the documents are checked and stored twice, and both banks must send requests to the third party when they need access to the documents. Unlike a sharing based solution, the burden on the customer remains the same, with them being required to provide the same documents to the central service for both banks. The cost and duplication of checks also remains. The figure also shows the extent to which the third party becomes a necessary part of the bank's day to day business once this model has been adopted. This means the bank risking being over reliant on the third party and has a single point of failures for KYC processes.

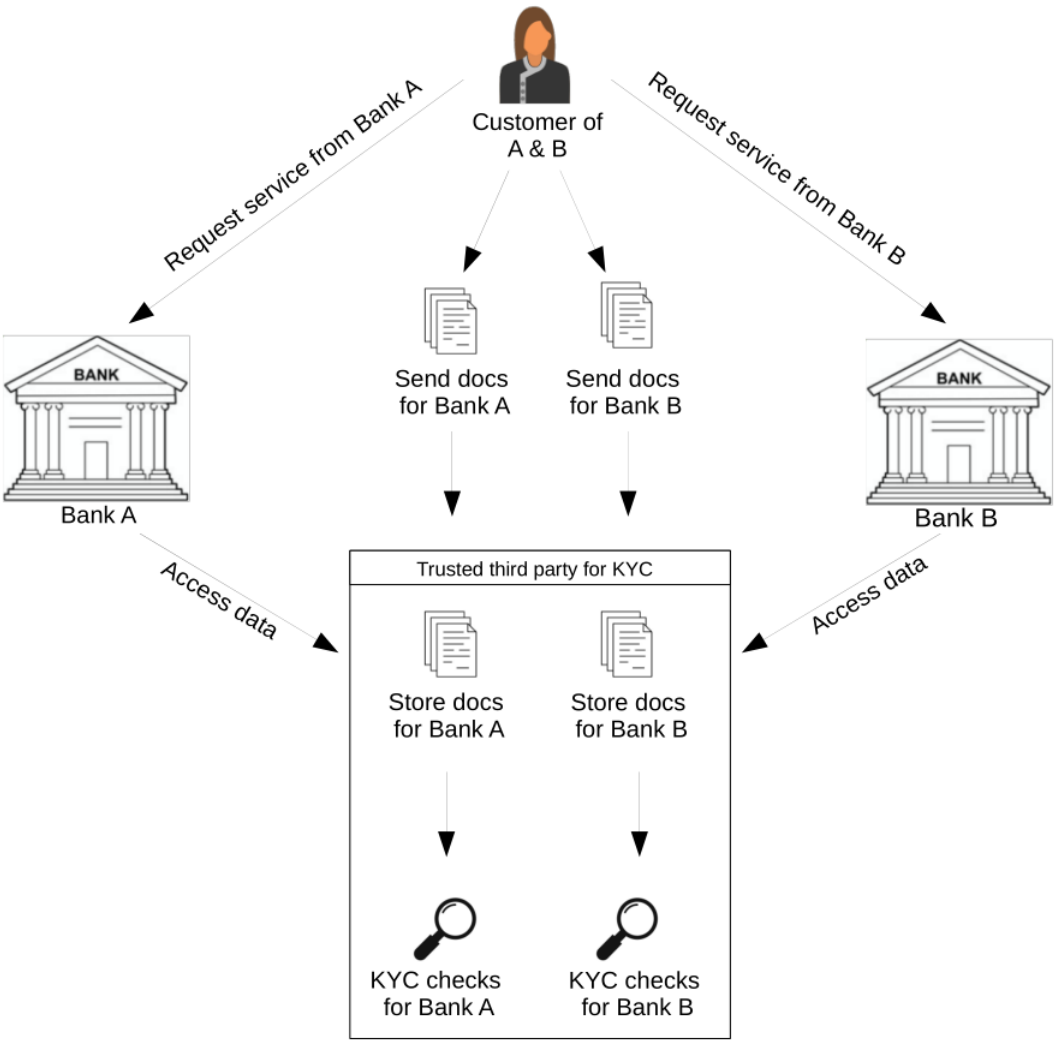


**Figure 2.1** Interactions in a fully outsourced KYC System

## 2.4 The Problem of Repetition and Manual Work

At present, KYC processes are carried out by each bank for each customer, with nothing shared between banks. Banks often share customers, private persons may have a relationship with multiple banks due to them shopping around for the best deal on a credit card, loan, or mortgage, for example. In the case of business customers, they often hold accounts with multiple banks in order to avoid chargers incurred when receiving or making payments between banks. As has been noted already in section 1.2.1, the complexity, and therefore cost, of compliance increases, depending on the client: from a national of the country in which the bank is located with the lowest complexity, up to multinational corporates, with the highest. However, KYC processes must be carried out for all customers. This leads to a large individual cost for each bank, even though they are dealing with the same customers and the same documents. Therefore if a customer is shared between at least two banks then costs can be saved through the sharing of information that has been collected already by one bank. Figure 2.2 shows a hypothetical 50% overlap in the customers between three banks. The intersections represent shared customers where savings could be made by sharing, rather than repeating document collection.

Sharing of KYC is able to have a greater positive impact on costs the more shared customers there are. The up shot of this is that the more banks there are in the data sharing system, the higher the likelihood that a customer's data can be shared. However, it is also possible to reduce the burden of on-boarding a customer whose data is not already shareable within the system.

Repetition is also an issue when dealing with trusted third parties; the issues a bank might face with centralised solutions and trusted third parties are detailed in section 2.4. However, sometimes banks are legally required to use third parties for certain services. Name list checking is one such requirement. All banks have a duty to check with whom they do business, which includes checking that potentially customers are not trading with black

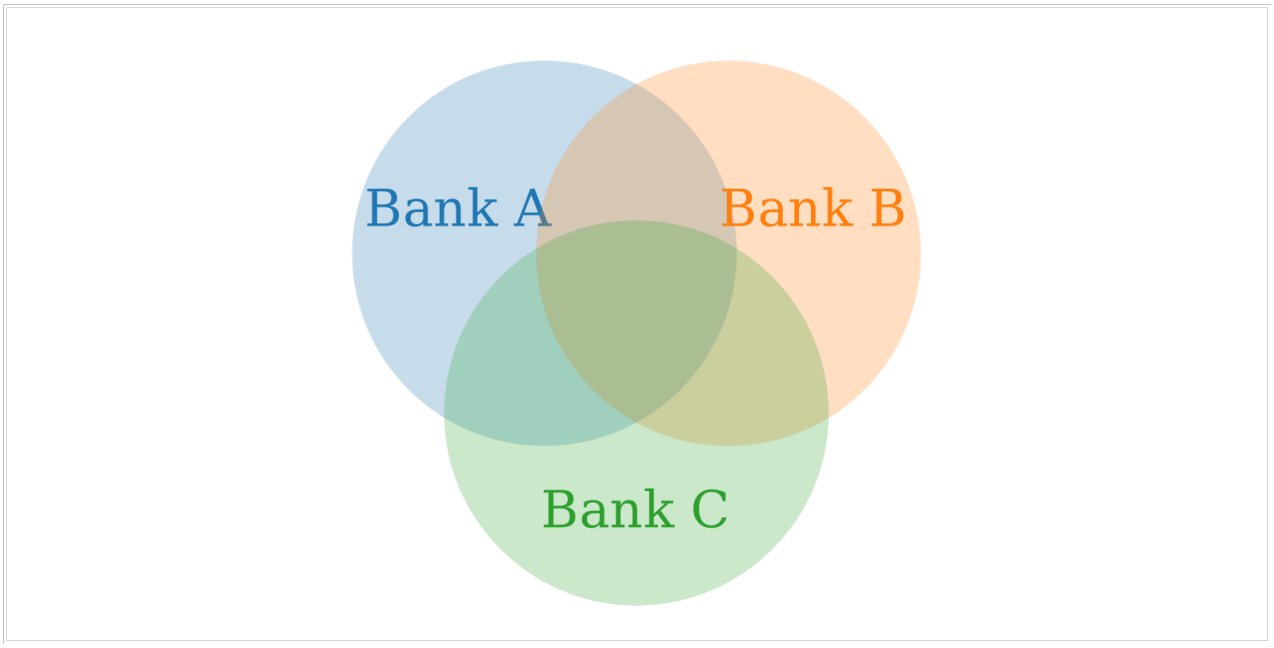


**Figure 2.2** 50% Overlap in customer base between three banks

listed countries, or involved in terrorism, or the funding there of. In order to meet their checking requirements banks must query name list services for each of their clients. They are mandated to check more than one service to ensure the thoroughness of checking.

Here there can be seen more repetition of KYC processes between banks. Name list checking is an automated process, with banks making an API call to the service and receiving a response in seconds. Nevertheless, the same API calls are made by different banks for the same customer, resulting in a duplication of work across banks. As each bank must pay for access to the API, and possibly for individual calls, a financial cost is incurred. Enabling the results of name list checking to be shared from one bank to another would form part of a disruptive KYC business model, with the banks standing to save money, and the checking services standing to lose out. Whether this can be done depends on the legality or receiving checks from other banks. However, it is technically possible to use hashing and/or digital signatures to prove that a name list check is authentic and has not been changed.

# Chapter Three

# State of the Art

> "It is in the laws of a commonwealth, as in the laws of gaming: Whatsoever the gamesters all agree on, is injustice to none of them."
>
> Thomas Hobbes - Leviathan

This chapter looks at the work done in the blockchain area and the current state of the art. It is split into 4 subsections. Section 3.1 looks at the first generation of blockchains and their functionality and limitations. Section 3.2 looks at the second generation, the advent of smart contracts, and the work done in terms of performance and security. Section 3.3 focuses on work done applying blockchain technology and smart contracts to new areas and fields. Section 3.4 details the gaps in the current literature in terms of a system that meets the needs of this work's use case. Lastly, section 3.5 looks at work done in the area of data pricing, and assesses to what extent the existing literature is a good fit for the KYC data sharing marketplace.

## 3.1 First Generation

The original blockchain was Bitcoin, detailed in a white paper authored by the still unknown person behind the pseudonym Satoshi Nakamoto in 2008 [72]. Bitcoin was originally created

with the sole purpose of allowing people to transact financially without the need for any third party, namely, banks. To achieve this goal, the paper details the various components of the chain, the linking blocks with hashes, and achieving consensus through PoW that many systems have since either utilised or adapted.

Litecoin is one such system; it was built using the Bitcoin code base, and like Bitcoin aims to provide a system for financial transactions without any trusted third party involved. Although the developers never produced a white paper it, like Bitcoin, is open source. Litecoin's github repository can be found at [60], here it can be seen that the Litecoin repository was originally forked from Bitcoin's [8]. Litecoin's major differences include the change in PoW algorithm to scrypt, a memory-hard algorithm which somewhat mitigates the hardware arms race cause by the creation of specialist hardware for solving Bitoin's PoW puzzle, which uses the SHA-256 hashing algorithm [73]. The original scrypt algorithm is detailed in [81], with modifications being used by Litecoin and other cryptocurrencies.

As Bitcoin development has continued, it has come to include a limited scripting language known as Script, that allow for more complex and secure transactions. Script is not Turing complete by design to prevent Denial Of Service (DOS) attacks against the system. Script, along with people's creativity in finding other uses for the system, led to some of the early non-currency based uses of blockchain. Notably for the topic at hand, these efforts include a Bitcoin based access control system [65]. Use of the Bitcoin public chain allows for the implementation of basic access control measures. However, the limited capabilities of Script, coupled with the high cost and low speed of transactions (10 minute block confirmation time) make it less than ideal for a lot of scenarios. Utilising a public chain allows for operation in the lowest trust scenarios as the public chain is the hardest to control and is globally visible. However, for KYC data sharing, the interactions should not be publicly visible. There follows below a discussion of the second generation of blockchains and the advances they made.

## 3.2 Second Generation

The major advancement in blockchain technology that characterises the second generation is smart contracts. Although Bitcoin has its intentionally limited Script, subsequent technologies took different approaches to the DOS problem and in doing so, were able to allow for the inclusion of loops as well as other common features of general purpose programming languages.

The first system to achieve global adoption of smart contracts was the Ethereum blockchain. The major driver of the second generation of blockchain technology, Ethereum is explained in great detail in its yellow paper [106]. Instead of removing features like loops to prevent malicious actors from sending transactions that would execute indefinitely, Ethereum has those wishing to launch or run a smart contract pay for the execution. Ethereum smart contracts are launched by users, and stored under their own address on-chain. When a user wishes to execute a smart contract they send a transaction to that address. Both launching and executing smart contract uses 'gas' which is paid for in Ethereum's native currency, ether. There is also a hard gas limit for transactions, which cannot be exceeded. As such lengthy execution is prohibited by the cost of doing so, and by a hard limit. It should be noted that users pay even if the execution fails.

The on-chain code that makes up a smart contract in Ethereum is called bytecode. Bytecode can be compiled to from a number of different languages. By far the most well used is Solidity; a object oriented language that looks similar to JavaScript (JS) [24]. A second language, Vyper, is currently in beta and is most similar to Python. It aims to be simple, secure, and auditable [25]. The ability to code with multiple general purpose language features led to an explosion in the number of possible applications, including in the areas of data sharing and KYC.

What makes Ethereum still more attractive for the proposed system is the fact that it is built with the ability to be run as a private permissioned blockchain, and provides

ready-to-use PoA consensus called Clique, which is much faster and less computationally expensive than the older PoW. PoA is well suited to a private chain environment such as that required by the proposed system. The documentation for the Golang [41] ethereum implementation, known as go-ethereum (geth), of PoA consensus algorithm can be found in tthe documentation for Geth [26].

Under Hyperledger as an umbrella project there are currently 10 distributed ledger systems [47]. One of these systems is Hyperledger Fabric, which is an open source project that provides a framework for building distributed ledgers [48]. Fabric can be used to build a private distributed ledger, it is modular by design and requires that each feature is 'plugged in' to build a system that meets the users' needs [46].

### Contract Performance

Given the prevalence of smart contracts in Ethereum, various projects and papers have focused on Ethereum bytecode. They fall into two main categories, those dealing with the monetary cost of executing smart contracts, and those dealing with contract security. In this section efforts focusing on performance are discussed.

In [16] the authors introduce GASPER, a tool which analyses smart contract bytecode. It uses symbolic execution to identify optimisations that were missed by the solidity compiler. GASPER does not require contract source code to function and is able to identify a high number of contracts containing candidates for optimisation. Unfortunately no source code is provided for the tool. The authors build upon their work on GASPER in [15], presenting a tool called GasReducer which includes the capabilities of GASPER, and extends them with the capability to find many more costly behaviours in smart contracts.

More recently, in [66] the authors present two methods for working with bytecode to find the worst-case gas consumption for a given smart contract.

The authors of [13] propose a tool based on symbolic execution which aims to find contracts which are inefficient in terms of gas, which is to say, contracts which are unnecessarily

expensive to execute.

With Pontiveros as first author, in [82] we approached reduction of chain size from a different angle, and treated it as a compression problem. By finding commonly occurring chunks of bytecode in Ethereum smart contracts we proposed to replace repeated code with points. Doing so achieves space savings of up to 75% in the dataset of contract bytecode.

The authors of [88] detail an orchestration framework designed to enable networked blockchain experimentation. The tool detailed in the paper aims to enable resource provisioning and blockchain configuration over the grid'5000 testbed platform.

### Contract Security

Ethereum has seen a number of high profile attacks on smart contracts which were key motivating factors for a lot of the security focused studies detailed below. The most famous attack to date is the Decentralised Autonomous Organisation (DAO) attack [31]. While not a flaw with Ethereum itself, the attack was possible due to problems with the DAO smart contract. The DAO was designed to act as an on-chain venture capital fund, with users putting in ether and voting on what should be funded. In brief, an attacker was able to make recursive calls to withdraw ether from the contract. As the contract sent the money before updating the user's balance the attacker was able to withdraw the equivalent of $70 million. The attack ultimately resulted in Ethereum splitting in two, leading to the current state of affairs where both Ethereum And Ethereum Classic public-chains in operation.

The second high profile attack is known as the Parity wallet hack [21]. Parity multi-signature wallets relied on a 'library' in the form of another a contract which they made calls to in order to execute its code. Unfortunately the library contract was coded in such a way as to allow anyone that called its initialisation function to take ownership of it, which is exactly what someone did. The contract's kill function removes the contract from the current and future states of the Ethereum chain. It can only be called by the contract's owner. The contract's new, self appointed, owner called the kill function, and in doing so caused wallets

holding 513,774.16 ether to stop functioning, meaning those wallet owners had in effect lost their money. The parity and DAO hacks are frequently used as benchmarks for tools aiming to detect security flaws in smart contracts.

Securify is presented in [99]. It operates on bytecode and carries out symbolic analysis before checking compliance and violation patterns to identify potential security flaws in contracts. The project is open source and was supported, among others, by the Ethereum Foundation [30].

Mithril classic is a security analysis tool for Ethereum smart contracts which makes use of concolic testing, taint analysis and control flow. It operates on bytecode and is capable of producing a visual representation of a contract, splitting it into functions and showing control flow paths by looking at the `JUMP` and `JUMPI` opcodes. It can be found at [17].

Vandal is another security analysis tool for Ethereum smart contracts. It converts bytecode into semantic logic relations and boasts a 4.5 second average analysis time [9]. The project is open source [102].

In [42] the authors present MadMax, a tool that uses static analysis of bytecode to find vulnerabilities caused by out-of-gas exceptions. They utilise control flow analysis to decompiler contracts and declarative program-structure queries. The tool operates exclusively on bytecode.

Oyente is a high profile tool that uses symbolic execution to analyse bytecode for security flaws. It is capable of finding a number of bugs, including the extremely costly DAO bug[64]. The project is open source [69].

In [61] Liu et al. present the ReGuard tool which finds re-entrancy bugs in Ethereum smart contracts through the use of fuzzing and analysing the resulting runtime traces. The authors were able to use the tool to find previously undiscovered bugs in smart contracts.

Similarly, in [49] Jiang et al. present a tool for fuzzing Ethereum smart contracts called ContractFuzzer. They also analyse runtime traces; the authors were able to fuzz 6991 contracts and found a claimed 459 vulnerabilities.

On the other hand, in [50] Kalra et al. take a different approach, aiming to verify the correctness and fairness of contracts through the use of symbolic model checking, augmented with abstract interpretation and constrained horn clause checking. They present a prototype tool tool called Zeus; after analysing 22.4k smart contracts the authors claim that 94.6% of contracts are vulnerable.

[14] details a tool called SODA with a broader scope. The authors aim to provide a framework that not only allows users to develop apps capable of detecting specific types of attack, but also provides an interface with which apps can be created. The tool is capable of operating on any chain that utilises the Ethereum Virtual Machine (EVM).

EthIR uses Oyente's CFGs to produce rule-based representations of contracts. In the authors' words this "enables the application of (existing) high-level analyses to infer properties of EVM code" [4].

Rattle is an open source project which converts bytecode to a CFG, then into an SSA form. The idea being that a contract's operations are much easier to see in the SSA form, which omits all stack operations [97].

In [32], with Torres as first author, we introduce a tool called Ægis which uses encoded patterns which define malicious transaction and the exploitation of loopholes and bugs. Unlike other approaches Ægis is designed to act in real-time on already deployed contracts, defending users and contracts against attacks after launch.

### 3.2.1 Chain Performance

Performance of blockchain systems is an issue which has attracted a lot of attention given the need to form consensus and the append-only nature of chains. Various efforts have aimed to optimise blockchain technologies, aiming to either increase the operating speed of a blockchain, or reduce the storage space it requires.

Some of this work has been done directly to the Ethereum software itself, such as the

implementation of fast sync, which allows nodes to download and validate the chain more rapidly, at the cost of not fully validating older blocks. Fast sync is well described in [93].

EIP 170 sets a hard limit on the number of bytes that a contract can be [10], although primarily done to protect against attacks that could slow down nodes, it also helps reduce the growth of on-chain data by limiting contract size.

Various attempts to improve chain efficiency involve moving processes off-chain in order to be able to process things faster, and to reduce the amount of data that is added to the main-chain. The light clients mentioned in section 2.1 fall into this category. The Bitcoin Lightning Network is a system designed to allow for faster transactions by moving work off of the main-chain [5]. Participants can conduct transactions off-chain and only submit the start and conclusion of their transactions to the main-chain, thus reducing its size by requiring it to permanently record less transaction data. Performance is improved as transactions can be carried out more rapidly off-chain. Plasma [6] is being developed for Ethereum; it also moves data and processes off-chain. It aims to store transactions and smart contracts, or parts thereof, off the main-chain in order to reduce the rate of its growth and allow for faster transactions.

Ethereum account balances, runtime contract code and contract storage data are stored as part of the state, which is stored as a hash tree. Storing each full state can consume a large amount of space, as such regular nodes store only the current state. The state is continually pruned to remove data which is no longer relevant. For example, old account balances can be removed from it. However, only states are removed, the contents of the other trees whose root hashes form part of the block header are retained. Notably for this project, this includes the transactions in a block. A good description of pruning can be found in Vitalik Buterin's blog post on the subject [11].

Storing files on-chain is inefficient and can rapidly increase chain size. In the Filecoin system [54] one can pay to have files stored in a distributed way, using the hash of the file stored on the blockchain to provide assurance for the integrity of file when retrieving it. Here

it can be seen that data is stored off-chain so it can be removed when it is no longer needed, with only an identifier stored on-chain. Filecoin utilises the decentralised database system IPFS for its off-chain file storage. IPFS is looked at in more detail in chapter 4, however it is worth noting that IPFS uses hashes to identify files [5]. This provides the highly useful property of being able to verify a file by checking its hash against the name (hash) of the file you were looking for. The equality of the two acts as proof the right file was received. Filecoin makes minimal use for on-chain storage.

Ethereum's developers are already working on a decentralised system similar to IPFS for off-chain storage called SWARM [36]. This will provide Ethereum with a native method of off-chain data transfer. Swarm is still under heavy development. The Solidity compiler appends a Swarm hash to a contract's metadata to allow for efficient distribution and verification.

## 3.3 A Multitude of Uses

Various academic studies exist that utilise blockchain for access control. In this section a closer look at these studies is taken; they are summarised in terms of their applicability to this project's use case in table 3.1.

In [59] Liang et al. propose a system which uses a private permissioned blockchain to record user-given permissions to access medical data collected by "mobile healthcare applications". The approach has a real world application with insurance companies being given access to the data in order to be able to better price their services. However, the use of cloud storage for the collected data runs contrary to the requirement of the partner banks that they retain all their own data in-house.

Xia et al. take a similar approach in using a permissioned blockchain to provide access control for medical data stored in the cloud. They highlight the fact that invited, verified users are accountable [107].

The use case for [89] also comes from the healthcare industry. Although comprehensive in the types of results that can be achieved from queries, there is a lack of security in terms of potentially leaking private information, something the authors acknowledge in the paper.

Zyskind et al. provide an early example of blockchain-based access control for personal data (2015), which stores permissions on-chain and suggests a centralised third party for storage of personal data. The also provide a formal definition of the protocols they created for access control [109].

In [65] Maesa et al. detail a system for granting and revoking access to data based on the Bitcoin blockchain. They make use of the public Bitcoin chain to store permissions. This comes with the disadvantage, for the use case of KYC in the banking sector, of being slower and more expensive, as well as transactions being publicly visible.

In [98] Troung et al. detail a system which they claim is a GDPR compliant, blockchain-based system for data management. Like the solution proposed in this thesis, they leverage the immutability of the blockchain to record permissions and highlight the fact that the ledger allows supervisory authorities to check on the validity of transaction, something that is required in the financial sector. They deploy a Hyperledger Fabric based system for access control. Where their solution differs to ours is their use of an "honest Resource Server", which acts as a trusted third party, and as such a potential point of over reliance for the banks.

Markus et al. provide a more complex Hyperledger Fabric based system designed to meet the needs of access control for enterprise applications. They allow end users to interact directly with the blockchain, utilising ring signatures to protect their identity. They also make use of split keys to protect private data. They use decentralised storage in form of Hadoop [67].

Bhaskaran et al. detail a Hyperledger Fabric based data sharing system in [6]. The authors present a system where all personal data is encrypted and stored on-chain. Access to the data is granted through the acquisition of the symmetric key used to encrypt the

personal data. Key acquisition is controlled by a smart contract that grants access based on permissions granted by users. Storing all data on-chain causes a chain to grow rapidly and precludes the possibility of removing any data. This could be problematic in the event that a key is lost or leaked.

In one of our previous papers we modified the IPFS client such that it checks against an Ethereum smart contract for the existence of a permission before serving a file. The smart contract holds the record of permissions and IPFS enforces the permission such that it would only make a file available if a permission to do so existed [90].

### 3.3.1 Blockchain for KYC

There is a strong indication from the financial sector that mutualisation is required and that financial institutions view it as beneficial for reduction of costs for KYC. In her 2017 banking and financial services report Lootsma details the potential cost reduction that can be provided by the utilisation of blockchain for KYC processes in the financial sector [62]. Moreover, the Digital Banking and FinTech Innovation Cluster, part of Luxembourg's own banker's association Association des Banques et Banquiers Luxembourg (ABBL) released the results of a 2019 survey which showed that KYC is the second highest in terms of interest for mutualisation, with 71.4% of responding financial institutions expressing a desire for this. Furthermore, KYC topped the list for areas of mutualisation that respondents believe constitute a competitive advantage for Luxembourg, and which have the potential to be successfully extended aboard. 71.4% of respondents agreed upon this [3].

Examples of the utilisation of blockchain for KYC and data sharing can be found at nation state level as well. In [91] Sullivan et al. detail Estonia's plans and use of blockchain technology in relation to their e-residency program. Interestingly in [44] a comprehensive review of blockchain technology in the banking sector is made. The authors conclude that KYC processes, among others, are one of the key opportunities for improvement through

the application of blockchain technology. They note the need for "extensive research and development" if banks are to adopt blockchain en masse. The authors of [68] arrive at similar conclusions; they state that banks are struggling with efficient KYC procedures, and that costs for on-boarding corporate clients are high, in addition to the processes also being poor from the client point of view. They conclude that blockchain can solve a number of KYC related challenges, but that blockchains are not without their own challenges. The importance of cooperating with different actors in the financial industry is highlighted. Such studies provides ample academic motivation for researching the application of blockchain to KYC.

In [71] Moyano et al. detail a distributed ledger based system to help reduce the costs of KYC processes for financial institutions and, similar to this project, their requirements come from an industry partner in the financial sector. They highlight the same issue as this thesis, namely KYC processes are replicated per bank. They also propose a distributed ledger with the banks as nodes where end-clients give permission to share their data between banks. Their assumptions differ to those made in this thesis. They assume all banks will be in the same nation and the system will be maintained by the national regulator. On the other hand, for this project the assumption is that a bank can be anywhere in the EU, with the system being maintained independently by the member banks, with national or international regulators being able to observe the system at will. Moyano et al. make use of centralised storage, under their use case it will be operated by the national regulator. This is the only KYC focused paper which discusses the pricing of data, with the authors detailing a model whereby remuneration is split between banks in the form of an on-chain currency which is tied to a fiat currency. There are a few major differences between the paper in question and this work. Firstly, in this work the scope is increased from national to international. Secondly the burden on the regulator is significantly lessened as they are not required to maintain the system or provide document storage. Thirdly, a range of pricing methods are considered, rather than only one. Lastly, it is worth noting that the requirements for this

project come from three industry partners, rather than one. This increase in perspectives helps to ensure the resulting system has applicability to the sector as a whole.

[80] builds on the work done in [71] and aims to improve upon it, evolving the solution to something closer to one that can be implemented by financial institutions. They rely on a query based method of data access detailed in [89].

In [7] Biryukov et al. present a system for proving identity for the purpose of KYC. It involves the user aiming to prove their identity directly in the process, and uses well detailed and defined proofs to ensure the system can be trusted. They note that GDPR is a strong motivating factor for the development and adoption of efficient KYC processes.

In [77] we detail a version of the KYC data sharing system developed for sharing KYC data between banks. An overview of the architecture and details of the ways in which the system is capable of reducing costs are given.

## 3.4 What is needed?

To summarise, there are a number of academic studies in the data sharing and KYC space. They take a range of different approaches and have various strengths and weaknesses. However, with regard to the particular scenario this thesis focuses on, reduction of KYC costs for the industry partner banks, it is possible to identify several gaps in the current work.

Firstly, none of the studies discussed provide solutions in which participants in a data sharing system are not dependent either wholly, or in part, on some third party for storage. The most common storage solution is a centralised repository which is trusted to store data securely, and respond to data requests based on the on-chain records. For the banks such dependence on a third party is undesirable as it leaves them exposed to increased costs. Therefore, there is a need for a solution which allows data to be shared with customer permissions, while also ensuring that DC's can participate in the system without having to change their internal storage solutions, meaning they should be able to continue to store all

| Name | Private | Open Source | Pros | Cons |
|---|---|---|---|---|
| Liang et al. 2017 | Y | N | Real world application with insurance companies | Centralised cloud storage |
| Xia et al. 2017 | Y | N | Real world use case with medical data | Centralised storage |
| Zyskind et al. 2015 | Y | N | Formalisation allows for implementation of the design | Centralised storage |
| Maesa et al. 2017 | N | N | Formalisation allows for implementation of the design | Bitcoin blockchain slow and expensive |
| Markus et al. 2019 | Y | N | Use of advanced cryptographic techniques to protect personal data | Partially outsourced private data storage |
| Moyano et al. 2017 | Y | N | Focus on real KYC issue of repetition | Centralised storage and lack of international applicability |
| Moyano et al. 2018 | Y | N | Can comprehensive data querying | Lack of data security |
| Bhaskaran et al. 2018 | Y | N | User actively involved in granting permission | All personal data stored encrypted on-chain |
| Biryukov et al. 2018 | Y | N | Formal proofs ensuring trust in the system | lack of real world use case |

**Table 3.1** Summary of blockchain-based data sharing systems proposed in academic studies

their data locally.

Some of the research efforts discussed have an encrypted, distributed storage solution. For the scenario at hand this would involve the banks holding data for one another, and therefore being dependent one another for storage. While the banks wish to cooperate to share data, they must retain all the data for all of their customers themselves. Banks are legally required to hold the data, and relying on a marketplace rival to do so is not advantageous for them. Moreover, it also increases the risk of the unavailability of data if the bank holding it suffers technical problems, or is offline.

Only one of the proposed systems discusses data pricing within the context of KYC. Given that data pricing is key to ensuring the system is equally attractive to banks of different sizes, with different amounts of data, it is worth focusing on in this respect. Moyano et al. [71] detail a method of pricing which involves diving the cost of data acquisition by the number of banks who want to use the data. However, their method of remuneration focuses on spreading the cost of KYC processes equally across the banks which have business with a given customer. This method may be more or less appealing to banks depending on the amount of data they have stored, as the cost is spread across banks, regardless of what cost they incurred when acquiring the documents. As such, further investigation into pricing models is necessary.

The same paper by Moyano et al. is the only KYC focused paper which has a set of real world requirements provided by an industry partner in the financial sector, this gives the work a lot of validity. However, as this project studies with three industry partners, rather than one, it is better positioned to ensure that the provided solution meets the needs of the sector as a whole, and is able to appeal to different sizes of institution with different customer bases and needs.

## 3.5 Pricing

A number of studies exist which look at how to price data. The majority of them focus on pricing data such that it can be purchased from the DO. This is not a good fit for this project as the focus for this project is on remuneration for incurred expenses between banks.

Li et al. discuss the cost of data acquisition for companies, and the cost of loss of privacy for the DO in [56]. Their focus is on compensating the DO based on how sensitive the data they share is. The trading of data is not restricted to any particular purpose, although companies must state how they intent to use the data they wish to purchase. The authors give formal definitions of cost functions based on the sensitivity of the data, with different functions for different types of data, the cost increases with the amount of data that is traded. It may be possible to repurpose some of the cost function work to help define prices based on cost of acquisition, rather than sensitivity. The work was written before GDPR came into force and certain details may need to reconsidered regarding the need for DO's to give permission for the use and sharing of their data. Lastly, the authors make use of a "two-part tariff" whereby there is a fixed cost for data acquisition and a variable charge which is dependent on the sensitivity of the data.

In [40] Gkatzelis et al. present their approach to pricing private data. As with Li et al., they focus on paying the DO directly for their private data. The authors have a strong focus on pricing data such that both buyers and sellers are properly incentivised to participate in the system, this is highly relevant for this project as one of the goals to be achieved through pricing is the incentivisation of different types of actors. The paper does not focus on a particular purpose for trading data. Interestingly, the authors define that a "market maker" should facilitate the trading of data, which is how this project's decentralised system functions. The authors give formally defined cost functions of both the linear and the concave type.

[39] provides a systematic review of literature on pricing of data "products" that are

traded on data marketplaces. The authors note that there is not yet a full understanding of how to price data in this way. They find that the use cases discussed in current studies fall into the categories of cities, business management, engineering, consumer, or linguistics. Since the financial sector and banking are not among the topics, it suggests further work is required in this field. Lastly, the authors note that they find a lack of real world testing in order to validate proposed models.

Liang et al. provide a survey focusing on big data which covers pricing and trading [58] of data, in particular they focus on the anticipated large scale data production of the Internet of Things. The authors highlight the need for the preservation of privacy when trading data. Pricing models are grouped into economic based pricing, and game theory based pricing. The former encompasses models based around supply and demand, and the latter are bargaining 'games'. The paper provides a number of models which could potentially be adapted to fit the project's marketplace. Prior studies are listed, each being assigned one of the pricing models, as well as the work's assumption about the market, whether the competition is high or low. In this context the project's marketplace would be categorised as low competition due to the fact that the entity which does the on-boarding or updating of documents is the one which is paid. According to the survey, low competition systems always adopt economic based pricing models, suggesting that models in this category would be a reasonable starting place when looking for the best model for the project.

In [57] the authors propose pricing models based in game theory. While game theory is likely to be less relevant to this project than an economic based model, the authors focus on pricing when the value of data is uncertain, which may be the case in a system with a variety of actors.

A game theory based pricing model is also presented in [104], where the goal is to reduce costs for the data purchaser. To do this the authors propose that DO's can be punished with negative payments if they give "unacceptably" high valuations of the worth of their private data. It is likely that such a method would not sit well with related legalisation in the EU.

Moreover, it seems unlikely that customers would be willing to enter into a system that might punish them if they do not agree to a price for their privacy within a range designed to keep the price low. When applied to banks, this clearly disincentives banks that would be primarily buyers.

In [55] the authors highlight the tendency for companies that provide services via the Internet to do so for free (in monetary terms) in exchange for access to the private data of the user. The work presents a query based pricing model where companies pay in order to make a query against private data and the resultant revenue is split between DO's whose data was accessed as compensation for their loss of privacy. This paper was published before GDPR and the method of pricing may prove problematic when one considers that companies must acquire the explicit consent of the DO to share data. However, the fact that users are willing to trade their data for free or convenient services suggests that uptake from the client side would be a problem for the system proposed in this thesis.

A query based pricing method is provided in [51]. The authors aim to allow for the pricing of any custom query against the data. The proposed method defines a way to automate the pricing providing it has some prior data of the price of 'a few views'. While the ability to price any kind of query provides great flexibility it is not required in a system where fairly standard sets of documents are being traded.

It should be noted that, [71] is discussed in section 3.4. This paper constitutes the only example, at the time of writing, that provides a method of pricing data between banks, and specifically for KYC. As was previously noted, the pricing model presented in the work focused on even distribution of costs, rather than the remuneration of the bank that originally incurred the costs.

Overall it can be seen that the majority of relevant studies around data pricing focus on paying the DO for their data. Whereas in this project the starting assumption is that the user is incentivised to use the system due to the convenience it brings them. More work is certainly required looking at scenarios where paying the DO is not the default assumption.

Moreover, pricing data specifically for KYC appears in only one prior work, and is not the main focus of the paper. Pricing data for a specific purpose is an area without much prior literature, especially for the case of KYC data.

## 3.6 Conclusion

In summary, there are two things missing in the current literature which the contributions of this thesis provide for. The first is the lack of banking sector-focused data sharing solutions. Moreover, existing solutions rely on third parties for storage, among other things. There is an industry need to be able to share and store data without the reliance on an external party to do so.

The second gap is related to pricing. Previous work in the area of pricing data largely focuses on paying DO's for their data, whereas under our system DO's must give their data as a legal requirement for using the services and the focus is on rewarding DP's which incur the cost of collecting and verifying documents. More specifically, suitably incentivising both those wishing to sell and those wishing to buy data is an open topic.

# Chapter Four

# Improving the Performance of Blockchain

"It's in reet fine fettle that"

Yorkshire Saying

## 4.1 Introduction

The work in this chapter can be found in the author's paper on the subject [78].

This chapter details a method of reducing the storage space required for the Ethereum blockchain which focuses on the storage space taken up by smart contract code. In terms of the KYC data sharing system, such methods of reducing space help ensure scalability into the future by providing ways to offload infrequently, or never, used data to specific nodes within the network, while retaining the ability to fully verify all such data if required.

Ethereum is set apart by its heavy use of smart contracts. They are compiled to bytecode from a source code language, most commonly Solidity, and stored on the blockchain as part of the transaction which adds them to the chain, this is known as a Contract Creation Transaction (CCTX). This means that all contract code is stored permanently on the chain including old and unused contracts.

Blockchains are append-only by nature, as data cannot be removed or changed without majority consensus. This is called a hardfork and is generally considered to be highly undesirable. As a consequence, full chains can become too large for average users to verify or store easily. Therefore ways to reduce the amount of data stored on the chain while retaining the assurances provided by a blockchain system are highly desirable.

IPFS is a program for the distributed storage of files across a peer-to-peer network, it stores files by splitting them into blocks and recording which blocks belong to which file. Each file is identified by its hash, and the user can check they have received the correct blocks by generating the hash of the file formed by the blocks and checking it against the hash they requested. Unrequested or unused files can be automatically removed from each participant's storage over time by IPFS's built-in garbage collector. Garbage collection handles local IPFS storage, however IPFS is designed such that files cannot be explicitly removed from the system as a whole.

This chapter proposes a system that leverages the inherent features of IPFS to reduce the size of the Ethereum blockchain. A method is detailed whereby code in Ethereum CCTX's is stored in IPFS, instead of on the blockchain directly. Currently, the bytecode required to create and run a contract is stored in a CCTX. By moving this code off-chain and storing a hash in the transaction instead, the size of the chain, and chain growth can be reduced. As IPFS identifies files by their hash, the inclusion of the hash of the code in the CCTX would allow for efficient and assured retrieval of the code from IPFS with the hash providing assurance that the retrieved code is correct and unchanged.

Ethereum contracts are stored in the state trie, meaning it is rare that contract code is retrieved from the transaction that created it. Nodes can download previous states from archive nodes. Therefore the code stored in CCTX's is bloating the chain. The data can be moved off-chain with little impact on performance and a significant impact on the size of chain data. IPFS is capable of removing unwanted files using garbage collection commands. `ipfs daemon -enable-gc` for automatic collection and `ipfs repo gc` to carry out removal

manually [53].

Users can prevent the local removal of a file under IPFS by pinning it, doing so tells IPFS never to remove it from local storage. Under the proposed system, the files stored on IPFS would be the contract data meaning a user can select which CCTX to store locally, if any. In the event that a user needs data for a CCTX that is not stored locally, any node can simply request the hash that is stored in the chain.

Ethereum has a number of full archive nodes that hold the data of all the previous states of the Ethereum blockchain. The functionality of such a node is not affected by the proposed system as it simply needs to pin all contracts to retain all the data that has been moved off-chain to continue acting as a full archive node. Morever, archive nodes, all ready hold all historical transactions in full so there is not change in storage requirements for them.

Standard nodes, when fast syncing, do not verify all transactions. Therefore replacing the code sent in transactions with hashes allows the nodes to request and store less data. Ethereum makes heavy use of Merkle trees, originally conceived in [70]. As each header contains the hash of the previous block and various root hashes of Merkle trees for a variety of relevant information, including transactions, the integrity of the code is assured. The hash in the CCTX cannot be changed without changing the hashes in the tree, and therefore the hashes of proceeding blocks as well. As the code associated with the hash cannot be changed without changing the hash, retrieval from IPFS is comparably secure to the code being stored directly on the chain. This not only reduces the storage space required for Ethereum end users but also lessens network traffic as nodes using fast sync to catch up need only be sent the hashes, and not the full code, in order to acquire the transaction root hash for each block.

The main contribution of this chapter is a method for reducing the size of stored Ethereum chain data by moving the bulk of contract creation data off-chain. To do so, some of the inherent properties of IPFS are leveraged. This reduces the amount of data sent over the Ethereum network as CCTX's can be stored without the full contract code.

The rest of this chapter is structured as follows: in section 4.2 related work done to reduce the size of blockchains is covered. Section 4.3 details the key features of IPFS and Ethereum in relation to this work, and how they are utilised. Section 4.4 covers how the dataset is constructed and how code replacement can be done. Lastly section 4.5 gives a detailed analysis of the experimentation and section 4.6 draws conclusion.

## 4.2 Background

Size and growth of public blockchains has been generally acknowledged as a problem. As append-only ledgers, the size of them will only ever increase. In recognition of this fact, various different methods have been used to reduce growth.

#### Nodes & syncing

Three different types of node exist in the Ethereum network:

- Full Archive Node: Stores every block and state in the entire chain. Fully validates all transactions. These nodes are generally only run by service providers such as `etherscan.io`.
- Standard Node (using fast sync): Downloads all blocks but checks only block hashes and receipts, not the details of each transaction. This allows it to skip downloading and/or storing each state in turn. Stores only the current state and discards old ones. Fast sync is well described in [93].
- Light Client Node: Stores only block headers and not full blocks. A light client is incapable of full independent verification and must request transactions it is interested in from other nodes in order to perform a minimal form of verification.

Of the above implementations, the latter two both aim to reduce the space required to store the chain by trading security for speed and size. Both rely on other nodes in the

network to store extra data should they need to refer to it at any point.

EIP 170 sets a hard limit on the number of bytes that a contract can be [10]. Although primarily done to protect against attacks that could slow down nodes, it also restricts the amount of data that can be added to the blockchain in one contract. In doing so it acts as a space saving mechanism.

### Moving data off-chain

Work has been done to reduce the size of Bitcoin's blockchain by moving a lot of transaction data off-chain. This idea has been realised in the form of the Bitcoin Lightning Network [84], wherein participants can conduct transactions off-chain and only submit the start and conclusion of their transactions to the main-chain, thus reducing its size by requiring it to permanently record less transaction data.

Plasma [83] is being developed for Ethereum. It also moves data off-chain. It aims to store transactions and smart contracts, or parts thereof, off the main-chain in order to reduce its growth rate.

### State tree pruning

Ethereum account balances, runtime contract code and contract storage data are stored as part of the state, which is stored as a tree. Each state consumes a large amount of space, as such regular nodes store only the current state. The state is continually pruned to remove data which is no longer relevant. For example, old account balances can be removed from it. However, only states are removed, the contents of the other trees whose root hashes form part of the block header are retained. Notably for this work this includes the transactions in a block. A good description of pruning can be found in Vitalik Buterin's blog post on the subject [11].

### Block data

Ethereum allows users to create and store smart contracts. They are created by announcing a transaction to the network with the code for the contract in the data field of the transaction. According to the Ethereum yellow paper the initialisation code is used only once to create the contract and then discarded [106]. However, as the root hash of the Merkle tree of transactions in a block forms part of the data stored in the block header, it is necessary to retain the contract creation data as part of the transaction. The transaction data is also required when any node wishes to fully verify historical transactions. Contract creation transactions are the target of this work's proposal to reduce the size of the Ethereum chain.

### Decentralised storage

Inherent properties of the Ethereum blockchain are leveraged to provide assurances about the integrity of data stored off-chain. This has been done for a different, but related purpose in the Filecoin system [54]. Under Filecoin one can pay to have files stored in a distributed way, using the hash of the file stored on the blockchain to provide assurance for the integrity of file when retrieving it.

### IPFS

IPFS uses hashes to identify files [5]. This provides the highly useful property of being able to verify a file by checking its hash against the name (hash) of the file that was requested. The equality of the two acts as proof the right file was received. Filecoin is build on top of IPFS.

### Swarm

Ethereum's developers are already working on a decentralised system similar to IPFS for off-chain storage called Swarm. It functions by using the Solidity compiler to append a Swarm

hash to a contract's metadata, which can be used to retrieve the contract. The main benefit over IPFS is the incentive system provided to users. Although financial incentives can be brought to IPFS, as done by Filecoin, it is likely that a natively integrated reward system would be good for user uptake. Swarm is still under development [36], however in the event of a full stable release, the proposed solution could be easily migrated from IPFS to Swarm.

## 4.3 System Architecture

This section details the key features of IPFS and Ethereum that are relevant to the proposed system. A describe of how each feature is leveraged or changed for the purpose of reducing the size of chain data is given.

### 4.3.1 Key Features of IPFS

In this section the key features of IPFS that this project makes use of are discussed in more detail. Full details of the IPFS system can be found in [5].

#### Garbage collection

Garbage collection removes the blocks of files which are unpinned. If there is a file that the user wishes to retain they can pin it, which marks the file as one not to be removed by IPFS. The user can pin/unpin a file using `ifps pin`. Files can be pinned when they are added to the IPFS using using `ipfs add pin` [**gls {ipfs}commands**].

In the proposed system the garbage collection feature is used to help reduce the storage space required for the Ethereum chain. In the case of regular or fast syncing clients the default will be for IPFS to store main-chain. This fits well with IPFS's paradigm of storing only what is specifically requested by the user, as well as with Ethereum's state pruning whereby information not immediately required can be removed. Moreover, an achieve node, wishing to hold all chain data would be configured to pin every CCTX data file.

In the proposed system the code moved off-chain is identified by its 32 byte SHA256 hash. Hashes would be stored for blocks of code $>$ 33 bytes in size, while code that is $<=$ 33 bytes could be stored directly on-chain, as no space would be saved by moving them to IPFS.

### Hashing

IPFS is capable of using different kinds of hashes to identify files with its multi-hash format. It consists of three fields which are as follows: The first is one byte in length and denotes the kind of hash (SHA2, SHA3, etc), the second field is also one byte and denotes the size of the hash. The third field contains the hash itself, its size defined by the value of the previous field.

### Block reuse

In previous work it has been demonstrated that contracts can have similar bytecode and similar functionality [76]. IPFS stores data in blocks which are up to 256 kb in size. Given the current maximum size of contracts, each will be stored in a single block. IPFS has the ability to assign the same block to multiple files if they share one or more identical blocks. In this scenario, if two smart contracts are identical then anyone storing both contracts would only need to store the block once. The chance of this is further increased as each contract is split into init code and runtime code. Identical contracts will result in an identical hash stored on the chain. As this indicates an identical instance of the code it should not be considered a problematic collision. It further increases the space saving when using IPFS.

### Pinning

Regular and light Ethereum nodes request information they do not store locally from archive nodes when they need it. Under the proposed system the data from CCTX's is held by archive nodes and can be requested when required. An archive node pins every file associated with a hash in a CCTX's data field. Any node can retrieve the bytecode associated with a hash

at will, without the current necessity of storing it permanently as part of a block. The role of archive nodes remains the same under the proposed system, and the data that is moved off-chain can be retrieved in the same way. Pinning also opens up an the possibility for a user to ensure the local existence and availability of a contract they deem to be important. For example, if one has a private wallet contract it would be possible to pin it to ensure its continued existence without relying on archive nodes. As long as the pinned local file exists it can be requested by other nodes.

### IPFS file propagation

By default IPFS does not store anything the user does not explicitly request it to. This will change the way data for contracts propagates through the Ethereum network. Under the current system, upon hearing about a transaction, a node will forward the transaction to its peer. In this way new transactions are flooded through the network. Under the proposed system flooding would still take place for transactions. However in the case of CCTX's, after becoming aware of such a transaction, nodes would carry out the additional step of requesting the contract bytecode from IPFS using the hash(es) in the CCTX. This involves adding the hash(es) identifying the code to their IPFS 'want list'. The node(s) from which the files can be obtained are discovered by searching the Distributed Hash Table (DHT). In the DHT file locations are stored based on the closeness of hashes at bit level. The significance of the DHT is discussed in the results in section 4.5.2. This is a shift from only listening for data to listening followed by requesting data. The model therefore shifts from push to push followed by pull. IPFS's default behaviour of storing only explicitly requested data allows for the minimisation of the local data storage per node. For code with a size $< 33$ bytes IPFS is not used and the original transaction propagation would take place.

**Table 4.1** Fields of an Ethereum CCTX

| CCTX | | | | | |
|---|---|---|---|---|---|
| nonce | gasPrice | gasLimit | ether | to | data |

### 4.3.2 Key Features of Ethereum Transactions

This section describes the key features of Ethereum transactions and how they are adapted for space saving under the proposed system.

#### CCTX's

Currently, Ethereum CCTX's consist of the fields shown in table 4.1.

The data field is the relevant one for the proposed system, it is an unlimited size bit array, in a CCTX it holds the code that a contract will have. Formally, it contains code that returns the contract's runtime code. This takes the form of init code followed by runtime code. Init code is discarded after contract creation, and is not used as part of the newly created contract. In the system, the hash of the file containing init or runtime code, as stored in IPFS, would be placed in the data field. Verifying nodes would then get the block(s) for each section of code through IPFS, check that the hash matches the one in the transaction (either for init or runtime code), and then verify the associated contract code.

#### Block headers & trees

Ethereum stores a number of root hashes of Merkle trees in the block header. Merkle trees have transactions as their leaves, whereby each transaction is hashed and trees are formed by combining and hashing inputs until the root is reached. One such tree is comprised of transactions. This means that block headers rely on the data of all transactions and full verification requires the full data of all transactions in block. Figure 4.1 shows an example of a transaction hash tree in Ethereum, with the data field being populated by the propose

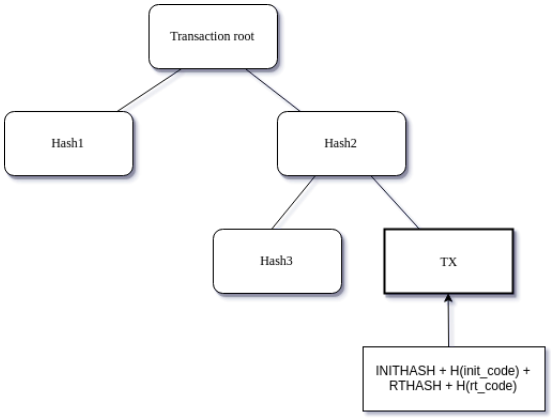


**Figure 4.1** Ethereum transaction tree including proposed opcodes and hashes

opcodes and hashes used to retrieve data from IPFS.

### Storing hashes

The most commonly used hash in IPFS is SHA-256. This hash is 32 bytes in length, most contracts are longer than this: between 514 and 1027 bytes according to the dataset. According to Ethereum's yellow paper init code is run once, then discarded [106]. Init code frequently exceeds a length of 32 bytes. It is therefore possible to retard the rate at which the blockchain increases in size if contract code is replaced with a hash that describes a file stored in IPFS. Contracts below 33 bytes continue to be sorted on the chain as there is no space to be saved by storing them off-chain. This necessitates that hashes representing init and runtime code must be prepended with an extra byte indicating whether the proceding data is code to be read directly or a hash of a file to be retrieved from IPFS.

### Contract Data Retrieval

In blockchain systems like Ethereum, weight increases trust in a transaction. Weight is the amount of valid blocks appended after the block containing the transaction in question. The more valid blocks, and therefore valid proofs of work, on top of the block containing a giving transaction, the less likely it becomes that the chain will fork below the given block. As a

result weight equates to trust. Therefore contracts that are less than $n$ blocks old could be pinned by default, where $n$ is defined as the number of blocks required to provide sufficient weight, meaning that no contract without sufficient weight can be removed. The contents of an IPFS file are assured in the same way regular transaction data is assured; by proof of work, or the weight of the blocks on top of a given block.

Optionally, blocks with sufficient weight could be trusted without ever retrieving the code and checking the hash. Fast syncing nodes do not fully check old transactions up until recent blocks at which point they begin to fully verify all transactions in a block. With the proposed solution they would not need to request the data for CCTX's that are not being fully checked. However, in the unlikely event that no archive nodes are able to provide the code associated with a given hash and assuming no user has the file pinned in IPFS a user could decide to trust the block based on its weight.

Section 4.3.1 describes how the interoperation of IPFS and Ethereum will change the way data propagates through the network in the case of CCTX's. This change also affects the way a user launching a contract on Ethereum is to behave. Currently a user announces a CCTX to the peers they are connected to, and can then disconnect from the network with a good level of confidence that the transaction will propagate through the rest of the network and be included in a block. However under the proposed system the user must also remain online long enough for their immediate peers to request and receive the data pertaining to the IPFS hash(es) in order for it to be propagated through the network as well. As the experimental results in section 4.5 show, this is likely to be no more than 541ms. As retrieval from IPFS will be an infrequent event, and block confirmation time is around 14 seconds, the proposed system will have little impact on the user sending the CCTX.

**Table 4.2** CCTX with init field holding proposed opcodes and hashes

| CCTX | | | | |
|---|---|---|---|---|
| nonce | gasPrice | gasLimit | to | `INITHASH` + $H(init\_code)$ + `RTHASH` + $H(rt\_code)$ |

### 4.3.3 Technical Summary

The proposed system involves adding contract data to IPFS unpinned, with a limited IPFS storage space so that contract creation data will be quickly removed from nodes' local storage after the transaction has been verified and the contract account has been created. IPFS acts as a cache for CCTX data which is not required for the general operation of the chain due to Ethereum holding the data it needs quick access to in the current state. The off-chain data can be retrieved through IPFS, from archive nodes as and when necessary. The data field of a CCTX would, in the event of code being stored off-chain, hold two opcodes which instruct the EVM to retrieve the code from IPFS by reading the 32 bytes following the opcode as the hash. Table 4.2 shows an Ethereum transaction with the data field holding the two opcodes and the corresponding hashes, with `INITHASH` and `RTHASH` being the opcodes and $H(x)$ being the hash of $x$.

Figure 4.2 shows the interactions between system components. When both sections of code have been replaced with hashes, the steps are as follows:

1. The EVM reads one of the proposed opcodes (`INITHASH`, `RTHASH`) and takes the 32 bytes after it as the file hash to collect from IPFS
2. The EVM requests the file through IPFS
3. IPFS returns the file matching the requested hash
4. The EVM verifies the hash of the file and creates a new contract in the standard way

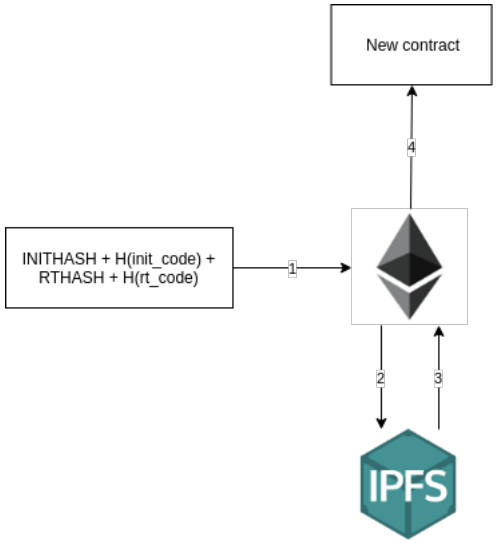


**Figure 4.2** Opcode, EVM and IPFS interactions

**Table 4.3** IPFS multihash format

| IPFS Multihash | | |
|---|---|---|
| fn code *(1 byte)* | length *(1 byte)* | hash digest *(variable len.)* |

IPFS records the hashes of files in its own multihash format. As the proposed system stores 32 byte SHA256 hashes on-chain, the EVM must build the multihash format in step 1 after getting the hash(es) from the CCTX. The format of the IPFS multihash can be seen in table 4.3. The EVM must build the multihash by appending two bytes to the SHA256 hash. The first field (fn code) denotes the type of hash, the second (length) dictates the number of subsequent bytes to be taken as the hash. IPFS works with base58 encoded hashes. The EVM must encode the constructed hash in base58. Once the multihash is constructed and encoded it can be requested from IPFS.

## 4.4 Methodology

In order to have a sample of recent CCTX's for the dataset a scraper is used to collect all CCTX's for a given number of blocks in Ethereum's main-chain from `etherscan.io`. The raw bytecode is translated into its equivalent opcode format. Each contract is split into init code and runtime code. This is done by finding the first instance of the 'STOP' opcode, with everything up to and including the 'STOP' being init code and everything after it being runtime code. The length of the two parts of each contract is calculated. As each opcode in Ethereum is one byte long, the number of opcodes is summed. 'PUSH' instructions take a subsequent number of bytes between 1 and 32 as a value to be pushed to the stack. As these values are a hard-coded part of the contract code, the sum of opcodes has the cumulative number of bytes denoted by 'PUSH' operations added to it. This forms the total size in bytes of each part of each contract. A contract's total size is the totals from the init and runtime parts summed together.

The savings made per contract are calculated by subtracting, from each of the two parts of a contract's creation code, one byte for the opcode that signifies the use of a hash, and the length of the hash. In simple terms: $codelength - 33bytes$. If the resulting value is $> 0$, and therefore a positive saving, it is added to the total savings. If it is $< 1$ then the saving is 0, as parts of a contract smaller than 33 bytes are left on the chain.

Two opcodes are proposed: `INITHASH` & `RTHASH` to act as instructions for when the EVM is to request files from IPFS. Under the proposed system the data field for a CCTX would have the format seen in table 4.4 for a CCTX where both init code and runtime code have a length $> 33$ bytes. The format can be adjusted for either code section being $< 33$ bytes by placing that section of code directly on the chain. By defining two new opcodes it is possible to maintain backwards compatibility with older contracts in Ethereum. If one of the new opcodes is encountered the EVM would acquire the code via IPFS, check the hash matches, and then continue execution on the retrieved bytecode. If the opcode is not encountered

the EVM will continue execution as normal. Therefore both new CCTX's with hashes, and those without will be able to function.

Formally, the construction of the data field of a CCTX is done according to the following: where *H(x)* is a function returning the SHA256 hash of $x$ and *len(y)* is a function returning the number of bytes making up the bytecode $y$. $ic$ and $rtc$ are init code and runtime code respectively. $+$ denotes the concatenation operation.

$$data = ic + rtc \tag{4.1}$$

Where:

$$ic = \begin{cases} INITHASH + H(ic), & \text{if } len(ic) > 33 \\ ic, & \text{otherwise} \end{cases} \tag{4.2}$$

and:

$$rtc = \begin{cases} RTHASH + H(rtc), & \text{if } len(rtc) > 33 \\ rtc, & \text{otherwise} \end{cases} \tag{4.3}$$

## 4.5 Experimental Results

The experimental results are split into two parts. Section 4.5.1 details results for replacing both code sections with hashes (two-hash solution), and for replacing only runtime code (single hash solution). The comparison of the two highlights that the lesser used and usually smaller init code section has a significant impact on-chain size. By leaving init code on-chain the number of IPFS retrievals the system needs to perform are halved, assuming both parts of code are $> 33$ bytes.

Section 4.5.2 looks at the impact of IPFS retrieval times and provides some analysis of file size based differences.

**Table 4.4** Structure of CCTX data field for using hashes

| INITHASH (1 byte) | init code (variable length) | RUNHASH (1 byte) | runtime code (variable length) |
|---|---|---|---|

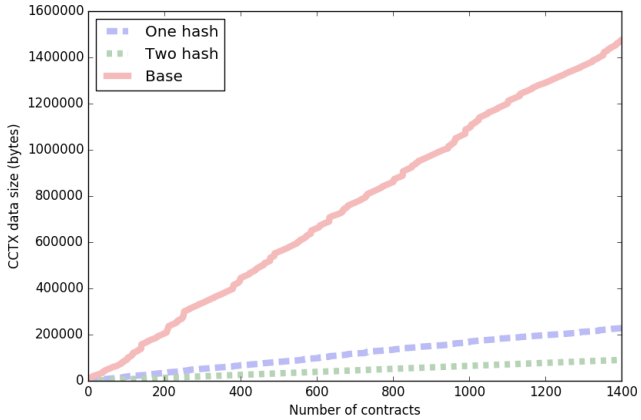


**Figure 4.3** Cumulative size of contract data in the transactions of the dataset

### 4.5.1 Code Size

The dataset contains all 1,403 CCTX's in the 1,048 blocks from the 5,000,000$^{\text{th}}$ block onwards. By counting the number of opcodes and the byte length of push values for both init and runtime code in each CCTX in the dataset it is possible to calculate the total size taken up by the code. It was found that runtime code was larger than init code for 1,345 of the CCTX's in the dataset. 1,027 contracts, which is 73.2% of the dataset, fall within the range of 514 to 1,236 bytes. Although far larger contracts exist, their occurrence is extremely rare. The single largest contract is 19,014 bytes in size. The mean average size of contracts is calculated to be 1,053 bytes, this contract size is used in the experimentation as one of the file sizes to test retrieval times. The largest init code is 2,150 bytes and the largest runtime code is 18,745 bytes. Mean init code size is 131 bytes and mean runtime code size is 922 bytes.

Table 4.5 shows the total size of each code segment for the base code, one and two-hash solutions. The runtime code makes up the majority of the base data, making the hashing

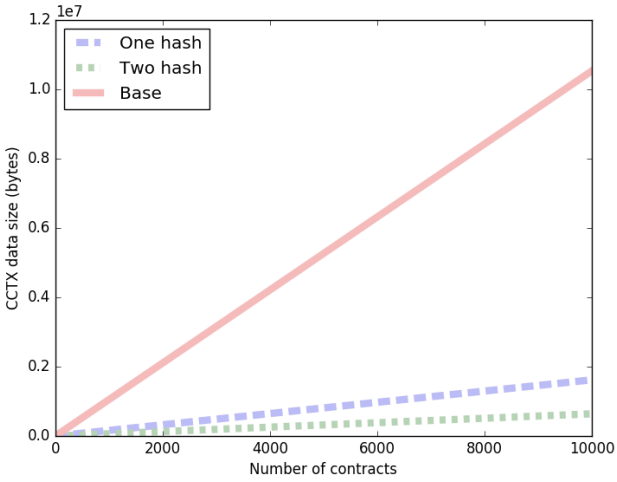


**Figure 4.4** Average size growth per CCTX

**Table 4.5** Comparison of total and average size of CCTX code in bytes

| | | **Base** | **Single Hash** | **Double Hash** |
|---|---|---|---|---|
| **Size** | init | 183,871 | 183,871 | 46,106 |
| | rt | 1,293,963 | 44,649 | 44,649 |
| | Total | 1,477,834 | 228,520 | 90,755 |
| **Reduction (%)** | | ∼ | 84.54 | 93.86 |
| **Avg. size per block** | | 1,410 | 218 | 87 |

solutions highly effective. However, the two-hash solution offers a larger reduction due to init code often being large enough to benefit from being replaced by hashes.

Figure 4.3 shows the cumulative size of CCTX data per CCTX in the dataset for the base data as well as the one and two-hash solutions. The two-hash solution shows a dramatically smaller rate of growth both the one-hash solution and the base size.

Figure 4.4 shows the estimated, cumulative size of CCTX data based on the mean average sizes of the CCTX data as it is in the dataset and for the one and two-hash replacements. These averages are summed cumulatively, up to 10,000 CCTX's. The two-hash solution provides the greater reduction due to it being common for the init code to be $> 33$ bytes.

### 4.5.2 Retrieval Times

In order to test the impact of IPFS retrieval on Ethereum test files were generated with specific sizes: 66, 1,053, 19,014 and 1,000,000 bytes. The first is the threshold at which code replacement takes place, the second is the mean size of contract bytecode from the dataset, the third is the size of the largest contract in the dataset. Finally, 1,000,000 is used as a stress test, it is larger than any smart contract is likely to be in the foreseeable future.

5 batches of 100 files of random data are generated for each of the file sizes specified above and added them to IPFS. The list of resulting IPFS hashes for each batch was passed to a server running a separate IPFS node. It attempted to retrieve each file and recorded the time taken to do so. The node serving the files was run in server mode to enforce external connections only, as is likely to the be the case in a real world scenario. This procedure was carried out for each of the 5 batches of each file size.

The mean retrieval time of each batch of test files is taken. All file sizes had a similar mean average retrieval time. The total spread is 62ms which suggests that IPFS is well suited to transferring the upper limit of present day contracts. The mean retrieval time for 1,000,000 byte files is only 42ms higher than that of 19,014 byte files, suggesting that

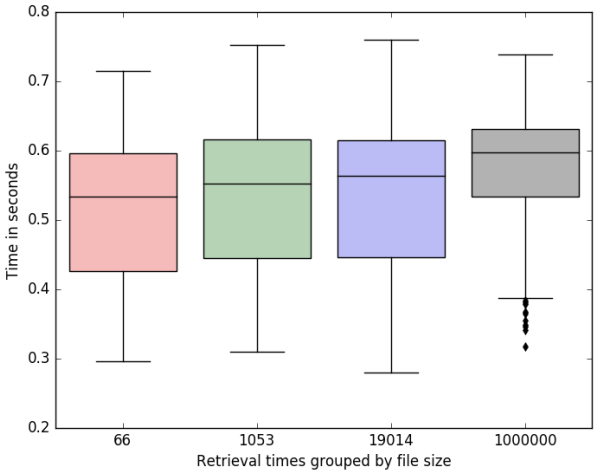


**Figure 4.5** Boxplot for retrieval times by file size

IPFS will be suitable even in the face of large increases in contract size. The mean average retrieval time for all files across all batches is 541ms.

The boxplot in figure 4.5 shows very similar medians and distributions between the first 3 file sizes. The rightmost plot, for 1,000,000 byte files, has a slightly higher median and a much smaller spread in the data. It is the only plot with any outliers.

The intersection for each pair of retrieval time histograms is calculated using the equation presented in [92] and shown here in formula 4.4.

$$int(T_x, T_y) = \frac{\sum_{i=1}^{n} min(T_x[i], T_y[i])}{\sum_{i=1}^{n} T_x[i]} \tag{4.4}$$

Where $T_x$ and $T_y$ are histograms of file retrieval times with $x$ and $y$ defining the file size. For example $T_{66}$ is the histogram of retrieval times for 66 byte files. $i$ denotes a particular bin in a histogram, with $T_x[i]$ being equal to the number of retrieval times in the $i^{th}$ bin. $min$ is a function that takes two bins $T_x[i]$ and $T_y[i]$ as arguments and returns the smallest of the two values. The result of $int(T_x, T_y)$ is the sum of the number of shared values in each bin as returned by the function $min$, divided by the number of values in the $T_x$ to normalise the results. The function assumes both histograms have the same number of bins with the same ranges.

Figure 4.7 shows that the plot for $T_{1,000,000}$ is left skewed. The plots for $T_{66}$, $T_{1,053}$ and $T_{19,014}$ all display a bimodal distribution. Table 4.6 shows the percentage of intersection of all pairs of histograms. $T_{66}$, $T_{1,053}$ and $T_{19,014}$ have a mean average intersection of 0.8809 and all 3 have intersections $> 0.85$. Each intersection with the $T_{1,000,000}$ is $< 0.79$, which is significantly lower than that the intersection between the other 3 histograms.

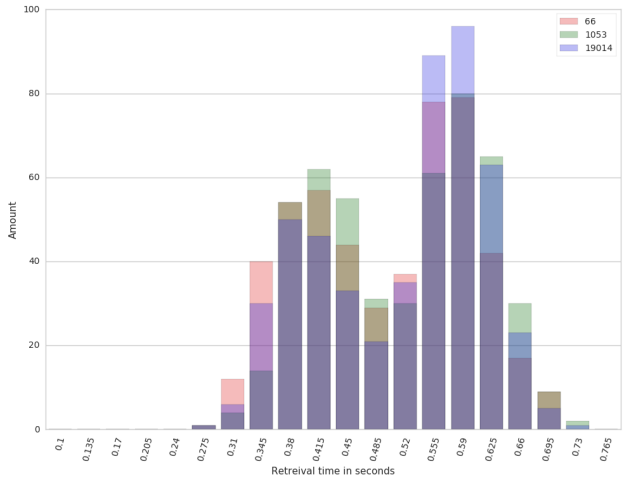


**Figure 4.6** Overlay of plots for $T_{66}$, $T_{1,053}$ and $T_{19,014}$

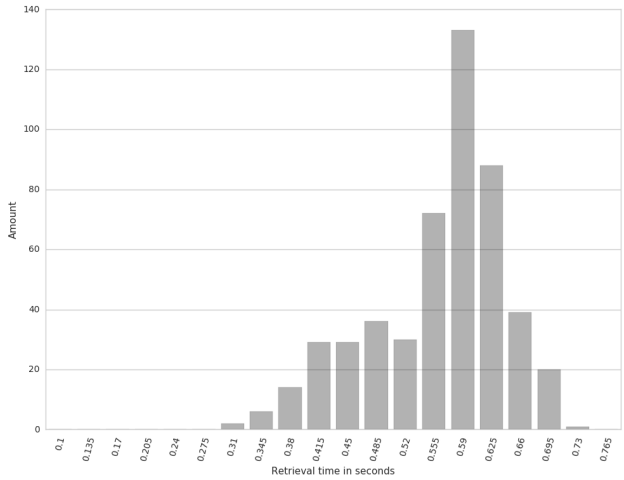


**Figure 4.7** Plot for $T_{1,000,000}$

Figure 4.6 shows the intersection of the 3 bimodal distributions. Given the higher intersection of these 3 and the different, left skewed distribution of $T_{1,000,000}$, it is conjectured that

**Table 4.6** Intersection of all pairs of bimodal latency distributions

| Pairing | % intersection |
|---|---|
| $T_{66} \cap T_{1,053}$ | 88.56 |
| $T_{66} \cap T_{19,014}$ | 88.96 |
| $T_{1,053} \cap T_{19,014}$ | 86.75 |
| $T_{66} \cap T_{1,000,000}$ | 71.89 |
| $T_{1,053} \cap T_{1,000,000}$ | 77.71 |
| $T_{19,014} \cap T_{1,000,000}$ | 78.51 |

the two peaks in the smaller 3 are due to IPFS's DHT. IPFS stores files based on hash. For smaller files the primary determining factor for retrieval time is the number of nodes in the DHT that have to be traversed. However, 1,000,000 byte files are large enough for internet bandwidth to become the primary determining factor over DHT look-up time. This explains the left skewed distribution for the largest file size in the experiments.

## 4.6 Conclusion

This chapter presents a method for reducing the size of Ethereum chain data and reducing the rate of growth for regular nodes by moving the code stored in CCTX's off-chain and replacing it with a smaller hash that allows for the retrieval of the code. This is achieved with minimum impact on system performance and ensures nodes retain the ability to fully verify transactions. As a positive side effect fast syncing nodes have to download less data. The reduction in size of the targeted data is roughly 90%. For the KYC data sharing system, such a reduction would aid performance by reducing growth in the case where the smart contracts it uses increase in number as more functionality is added, or when the system's smart contracts are frequently updated. In this case, the old and unused contract data can be stored off-chain for standard nodes.

All the code used for this work, including the scraper used to create the dataset and the code to carry out all tests, calculations and graphs is available in a github repository [74].

# Chapter Five

# A Decentralised Solution: Architecture and Implementation

> "It's not worth doing something unless someone, somewhere, would much rather you weren't doing it."
>
> Terry Pratchett

## 5.1 Introduction

The work in this chapter is drawn in part from [77], in which the system is detailed.

When a bank first on-boards a customer, it is responsible for gathering all the documents which are legally required for carrying out the necessary KYC checks. However, at present banks do not normally share their KYC documents, or the results of their KYC document verifications and checks on customer data. This is due to each bank being legally responsible for any data it holds, and the fact that banks are market place rivals. As such, there is a huge duplication of document acquisition and verification work carried out, with each bank acquiring and verifying the same documents for the same customer. From a client perspective, the lack of data sharing means that using the services of another bank requires all the necessary documents to be collected and presented to each bank. This is especially

onerous for more complex entities such as corporates, for which the amount of documentation required is often large and requires more regular updates. Therefore, banks have a need to exchange data in order to: reduce the cost of compliance, ease the burden on their customers, and ensure banks fully comply with KYC/GDPR legislation while sharing data. The end-client's need is to reduce the time spent on the provision of documents. In order for banks and their clients to use a data sharing system such as the one detailed in this work the end-client must be confident that all data will be secure and shared only at their request. This means a system with strong data access control and data privacy methods is necessary.

The system detailed is designed for banks in the EU. They are legally mandated to comply with KYC and AML legislation, and the recently introduced GDPR. As such, banks now have more compliance requirements than ever before. The focus is on two major issues surrounding KYC. Firstly, the time and expense incurred by carrying out the KYC processes necessary to comply with the legislative areas mentioned above. This is done by allowing banks to share customer data between one another. Through doing so the banks are able to reduce their costs and save time for their clients by introducing a mode of interaction between banks which did not previously exist. Secondly there is a focus on issues surrounding the privacy and security of KYC personal data, especially when transferring it between banks. GDPR requires that the end-client controls their own data; they control which entities have access to their data and the purposes for which such entities can use it. The end-client also has the right to request the deletion of their data, other legislation not withstanding. The system ensures GDPR compliance by storing explicit permissions to access customer data on the distributed private blockchain. Documents are never stored on-chain, in addition use of cryptographic techniques such as salted hashing ensures an end-client's request for deletion can always be complied with.

Banks have various different kinds of customers, all of which can benefit from the system. Customers are categorised as: natural persons, legal persons or corporate entities. KYC legislation applies to each of these categories. While GDPR focuses on the rights of natural

persons in relation to their data, the system can be used in the same manner, with the same assurances of security and privacy by legal persons and corporates to control the flow of any data sensitive data, and ensure their banking partners can access it. All three categories can ensure their data is accessible only to financial institutions with which they have business. Moreover, the banks can be sure that they are easy able to comply with KYC legislation in most cases as they can be notified when a document is updated by one of their clients. Banks are assured that the risk of a data breach is minimised due to the fact that all of their data remains stored in their internal encrypted storage, and that they have control over the data flow between their internal system and the KYC sharing system.

The system was built to provide cost reduction while ensuring security and privacy. It has a number of different data features which meet the requirements detailed in section 5.2. These include blockchain-based data access permissions to control the flow of data between custodians, to ensure end-client privacy and anonymity is maintained within the system. Pseudonymous (random) identifiers are used on-chain to protect privacy. Salted hashing is used where appropriate to allow for anonymisation of identifiers. Lastly, container technology is used to segregate the bank's databases from the data sharing system, all of a bank's data is retained by them individually.

The problem this work aims to solve with the system detailed in this work is directly defined by industry requirements. Work is done in close collaboration with the industry partners in ABBL. Within the ABBL there are three partner banks that provide the requirements. The banks range from European to global actors, and have areas of operation from the retail market to wealth management and corporate banking.

A recent study carried out in Luxembourg showed that regulatory compliance, KYC compliance, and standardisation hold the top three spots for processes banks believe it would be beneficial to mutualise [3]. In this work a P2P, blockchain-based system with the banks as peers, is proposed as a solution. It aims to enable data sharing between banks and address the security and privacy requirements of banks and their clients in order to reduce

the burden of compliance for all parties. As the system is built to be aligned with GDPR, the client is always in control of their data, granting explicit permissions for access.

The main contributions of this work are as follows: 1) A blockchain-based data sharing system, designed to meet industry requirements including reduction of costs via data sharing and legal compliance. 2) The system provides a storage solution that is minimally disruptive to a bank's operations and allows them to fully retain their data. 3) Lastly, the results of testing showing how the blockchain system performs under load.

The system makes use of a number of different concepts and technologies, including a private, permissioned blockchain for access control, and containerisation as a method of data segregation. The ledger acts as a record of end-client permissions and enables the banks to share data between one another due to the existence of the, permanent on-chain record. Container technology allows the system to be deployed on almost any operating system and hardware combination while ensuring the same standard of data security, without directly touching a bank's internal systems.

The rest of this work is structured as follows; section 5.2 details the requirements of the use case as guided by the real world scenario and industry partners. Section 4.2 discusses the major blockchain systems and the various efforts that have applied them in order to create data sharing systems. Section 5.3 details the system and its functionality. Section 5.4 details the technical implementation of the system. Section 5.5 details the tests carried out and interpret the results of these tests. Finally, section 6.5 provides the conclusion.

## 5.2 Requirements

This section details each of the requirements by identifying the needs of the banks, and then explains how the system aims to meet these needs. Each requirement is a result of the major challenges surrounding KYC.

**Reduction of costs:** The system is designed primarily for banks with operations within

the EU. With KYC, GDPR, and AML laws, banks have greater EU compliance requirements than ever before. Each bank is legally responsible for gathering documents and checking them for each customer. KYC Checks for natural persons include list checking for persons linked to terrorism and Politically Exposed Persons (PEP), and collection and verification of various documents depending on the client and the service they want. For legal persons and corporate entities the documentation can be large and varied, often with regular updates required. The documents banks hold must be kept up-to-date, they should always hold the current version of a passport, for example. Gathering and updating documents is an extremely costly process, especially for complex entities such as corporates. Banks have a need to reduce the costs, both financial and in terms of time, that stem from legislative compliance.

Data sharing is used to meet this need. Costs are reduced for both banks and their customers when data is shared between banks. Specifically, the customer need not present the documents, and the bank need not collect them manually. The primary function of the system is reduction of costs through data sharing. In order for this to take place there are a number of other important requirements that must be met, they are detailed below.

**Trust:** Each bank is legally responsible for the validity of the KYC documents they hold. Moreover, banks are marketplace rivals, making cooperation difficult. With this in mind, in order for banks to be able to exchange data between one another with confidence, data must be accurate, and providers must be accountable. The banks must be able to trust in the integrity of the data they receive. Customers must be able to trust that their data will only be shared with their permission. Trusted records must be kept, for permissions, audit and accountability. Furthermore, banks do not wish to rely on any single centralised third parties for the acquisition, storage, or transfer of data. Doing so exposes the bank to the risk of becoming heavily dependent on one entity, which could incur them great expense, or expose them to legal consequences if such an entity did not perform properly. As banks are market place rivals it is also inappropriate that any one bank should control the system or

the sharing of data.

As such, blockchain is utilised to act as a decentralised, immutable record of data access permissions, observed and agreed upon by all participants, to enable trust in the data access control and integrity of data. No single entity needs to be trusted in order for the chain to operation, and the banks do not have to fully trust one another.

**Data storage & flow:** Banks require that their customers' private data is always secure, and that if it is shared, it must be done selectively. Moreover, as a hard requirement of the industry partners, banks must retain all data about their clients, and not rely on each other, or third party solution for storage of data. Simply, the banks must be able to continue to store their own data locally.

The use of container technology ensures that the bank's internal, encrypted databases are segregated from the system that handles the sharing of data. The bank's existing storage system remains untouched, and the data for all their clients is stored within it. Data storage is explained in detail in section 5.3.2. Any data passed to the data sharing system must first be requested via an internal API call to the hosting bank. As such, banks are in control of the data they hold at all times. Moreover, the customer is given control over the flow of their data between custodians as data transfer cannot take place without the existence of their explicit permission.

**Data privacy & pseudonymisation:** As previously stated, banks must comply with the relevant EU legalisation. Care must be taken when using distributed ledgers as they are necessarily visible to all participants, in order to achieve consensus. It is vital that no personal documents are stored on the chain in order to ensure alignment with GDPR. Salted hashes are used to decouple the hash from the personal data used to generate it. Through doing so it becomes possible to store pseudonymous hashes on-chain, and still use the hash to prove data integrity. The salts are stored off-chain, allowing a bank to delete them and break the link between customer and hash, rendering the on-chain hash anonymous. The immutable on-chain record for access control ensures that the customer is in control of their

data.

**Ease of integration:** For the banks it can be very difficult and time consuming to install different languages, libraries and pieces of software due to internal security and auditing requirements. It can take a long time for each individual piece of software to be cleared for installation/integration into the existing systems. Containerisation is used in order to make it easy for banks to run the required software. Docker is used for containerisation, it provides a well recognised and supported system building and deploying container. As such, the only requirement is that banks install the Docker client software. This helps to ensure the requirement that the solution can be deployed as frictionlessly as possible, with minimum disruption to the bank's existing systems is met.

**Freedom from over reliance:** Having to depend on any third party for a particular service exposes the bank to the risk of dramatically increased costs, especially if the third party is a sole provider, or difficult to disengage with. Examples include list checking services, which banks are required by law to use, and outsourced KYC solutions. In case of the former the price can be steep simply because the banks must use the services. In the case of the latter, the bank is free to choose which, if any KYC service it uses. However, centralised KYC solutions hold the bank's data on their behalf. Once a large amount of data has been transferred to the service, or a large amount of new data has been acquired and stored by them, it becomes increasingly difficult in terms of cost and time to disengage with the service and move the data back in-house. Moreover, if document collection and verification were to be outsourced, the bank would be required to reinstate these services in-house in order to be able to disengage. The system is designed to ensure that banks are not overly reliant on any one entity. Data can often be provided by multiple banks, all data storage is local, and as each bank can trade the data they collect and verify, banks should retain their on-boarding abilities, while still reducing costs.

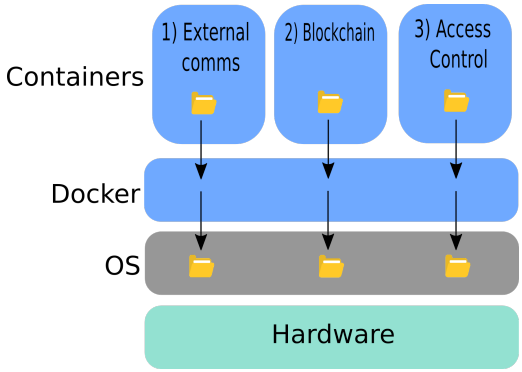


**Figure 5.1** Vertical slice view of containerisation and mapping of directories

## 5.3 Design

This section details the design of the system and how it meets the requirements listed in section 5.2. The following is discussed, the implementation and use of: containerisation technology, a private Ethereum blockchain, a smart contract, and API's.

### 5.3.1 Containerisation

One of the requirements is that integration should be as frictionless as possible. Therefore, the only requirement for the bank is that they install the single piece of software required to run the containers, and implement functionality for the predefined API for communication between the solution and the bank's internal systems. This is a large a reduction in work on the bank's part, compared to clearance, installation, and maintenance of various pieces of software and associated libraries etc.

As containers are transparent to their host, and under the control of the host, banks are free to inspect and test the container to their satisfaction in a sandbox environment before going live. They are also free to halt the operation of any container(s) should they feel the need to, thus meeting the requirement that banks are in control of their client's data at all times.

The system's functionality is spread across three containers, each of which houses a

logically separate part of system. This allows different functionalities to be logically separated out. Doing so allows for the system to be more modular, meaning different sections can be upgraded or changed independently of one another.

In terms of security requirements the use of containers allows for the implementation of data segregation; it is possible to separate data processing, storage, and transfer from one another. What containers can interact with, and the access they have to the host system is limited. An example of the three containers' separate functionality and access to the host system can be seen in figure 5.1. Moreover, the container responsible for data transfer must request data from the bank's internal IT systems via API calls, meaning the bank has the final say on whether a document should be supplied or not. The containers run by each bank can be seen in figure 5.2. The two large boxes each represent one bank, with boxes numbered 4 and 8 being each bank's internal IT system, respectively. The containers which comprise the system are boxes 1-3 for the first bank, and boxes 5-7 for the second bank. Figure 5.2 also shows the internal and external API based interactions for each container and the bank's internal system. The external communication channels between the two banks can be seen connecting the blockchain (2) and data access control (3) containers to their counter parts in the second bank, 6 and 7 respectively. The explanations below focus on the first bank, however as all banks have their own instance of each of the containers, the explanations apply to every participating financial institution.

### 5.3.2 Data Storage

All personal data is held by the banks themselves, in their pre-existing internal storage solutions. Each bank retains all the data for each of its customers. This was a hard requirement for the industry partners, as such the system detailed in this work facilitates the sharing of data, but never stores it. Such a system also meets the requirement that there is no reliance on a centralised third party. To do so could expose a bank to the risk of increasing costs.

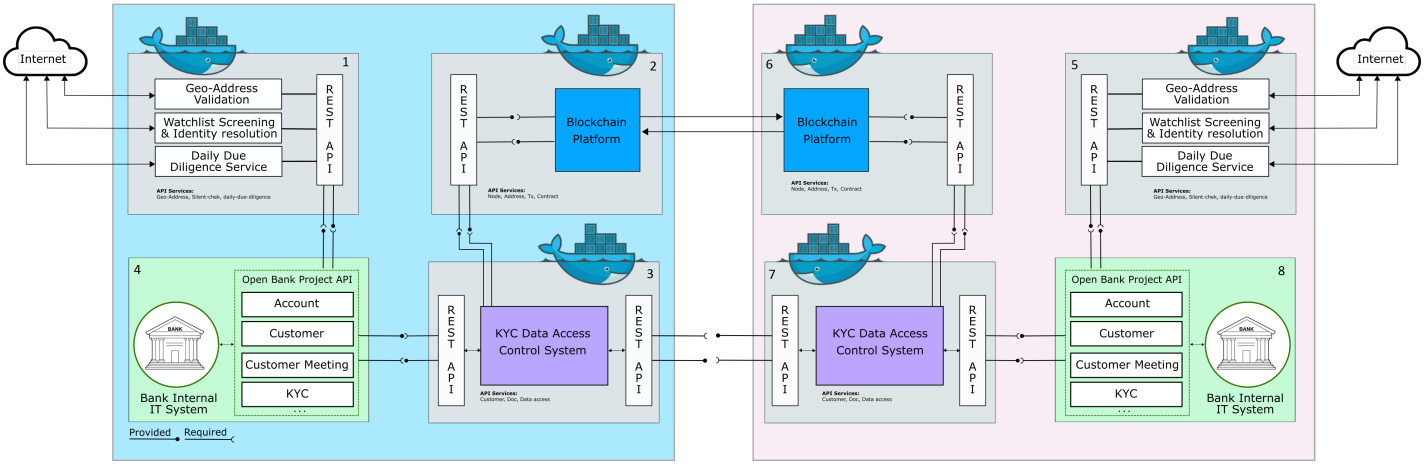


**Figure 5.2** System Architecture for two banks running the system

The data storage model of the system can be seen by way of and example in figure 5.3. It depicts the scenario where Alice is already a customer of bank A and wishes to become a customer of bank B. In step 1) she gives bank B permission to access the required documents. In step 2) bank B sends the request for the documents to the system, bank A sees the request is valid and can provide the data. In step 3) bank A retrieves Alice's documents from its local storage system and makes them available to bank B. Lastly in step 4) bank B receives the files from bank A and stores a copy of them in its own local storage. Alice is now a customer at both banks and the necessary KYC data acquisition has occurred, with each bank retaining full control and storage of the data pertaining to their customers.

The chain provides the banks with an audit trail as to who provided them with what data and when. This immutable audit trail ensures banks behave honestly with one another when sharing data. As the proposed system acts as a facilitator for data sharing between banks and does not aim to store their data, the banks requirement are met and they are able to participate in the system.

Due to GDPR it is not possible to store any personal documents on-chain. The on-chain identifiers take the form of randomly generated ID's for end-clients, and salted hashes for documents. Both the ID and the hash is designed to be separable from the end-client, that is to say, both can be made anonymous, in alignment with GDPR. In both cases each bank

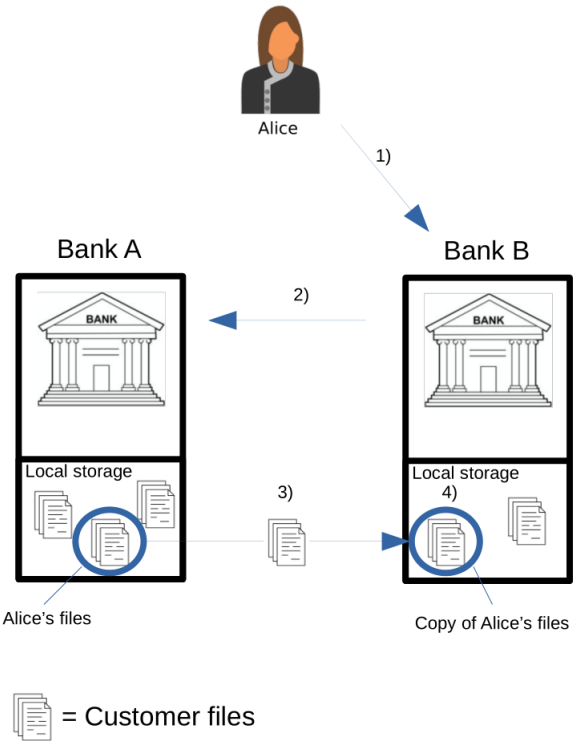


**Figure 5.3** High level overview of document acquisition process for a customer beginning a relationshp with their second bank

is required to hold a mapping in its own internal storage systems for its own clients, and only for its own clients. For each client's random on-chain ID, the bank holds the ID in their databases in such a way that it is associated with the client, as they would do for a phone number, for example. This ensures that only banks with which the client has business are aware of which on-chain ID belongs to the client. This ensures no personal information is indirectly leaked to banks with which the client does not have a relationship. If a client ends a relationship with a particular bank and makes a request for deletion of the data then the mapping between the client and their on-chain ID would be deleted. This breaks the link and re-anonymises that client in the eyes of the bank complying with the deletion request. This ensures GDPR compliance.

For salted hashes of documents the process is similar. The bank stores the document and associates the relevant salt with it. The hash cannot be generated without both the document and the salt. If the client requests deletion the bank deletes the document and

the hash. The on-chain hash has been re-anonymised from the bank's point of view as they cannot recreate it, ensuring GDPR compliance.

### 5.3.3 Data Flow

To give participating banks the highest possible levels of assurance about when and with which entities personal data can be shared the system is designed to be entirely pull based. In addition to data sharing being controlled by customers via the blockchain's record of permissions, data can only be accessed via API requests. The existence of a permission does not mean that the entity to which permission is given automatically receives the files. Rather, such an entity must make a request for the data. This ensures that the flow of data will not be unduly restricted and will be comprised of only what each bank needs.

Moreover, in keeping with the requirement that banks are always in control of their data, the data access control (DAC) container is only able to access data via API calls to the bank's internal system. As such, the bank has the final say on any data leaving their system. If they wish to double check something, or have a doubt about a particular request then they can withhold their response to it.

### 5.3.4 External Communications Container

Container 1 is responsible for data checks and automation that require communication with third parties. To help ensure data security, this is the only container with access to these communication channels. It requests and retrieves data from third parties used by the banks to check watchlists and valid address data. This container interacts with the bank's internal systems and not the other containers directly. This segregates the only part of the system that communicates with third parties from the containers handling access control permissions and data transfer. The bank can make calls to the container when they require any of the checks it is capable of carrying out. The bank has full control over the returned data in

terms of what is stored and how. The checks this container is capable of can be augmented to include any that the banks participating in the system agree upon. Here trust in the data is enhanced as any check handled by the container is carried out in the exact same way regardless of which bank does it.

### 5.3.5 Blockchain Platform Container

Container 2 holds the geth Ethereum client software. It has a white list consisting of the other blockchain containers in the system, and will only communicate with those nodes. The blockchain records include observed and agreed upon results of smart contract execution. The smart contract is used to record permissions and check whether each request for data is valid or not, based on the permissions. This means that both banks and customers can be sure that data is only shared when the permission to do so is given. In this way the flow of data between banks is controlled in order to meet trust and privacy requirements. Container 3 will not act upon a data request without first receiving the appropriate response from the blockchain container.

The blockchain container has no access to any documents, limiting the possibility of identifying information being made visible, and ensuring data segregation. As the blockchain software is containerised it can be swapped out for a different or updated blockchain software, should it be considered that one is more secure or faster in the future. At present the working prototype runs with both Ethereum (geth) and Hyperledger Fabric depending on which container is connected.

### 5.3.6 Data Access Control System Container

Finally, container 3 interacts with container 2; it also operates a white list. Upon receiving a data request, container 3 will wait for the response from 2 before making a call to the bank's internal systems (4), to request the necessary customer documents. Here one can see the

segregation of data from operations in order to maintain data security. As the flow of data is dictated by a pull model, rather than a push model, container 3 has no access to any data without the bank sending it. Container 3 cannot make a direct database query, to retrieve documents by itself. As such, the bank can analyse any request and ensure everything is in order before responding. If there is any doubt the bank simply needs to not send a response to the request and all data is secure. Moreover, the data is segregated from the other containers in the system as they have no way to request data. Containers only interact using their specific API's. Secure data management is ensured here as at each stage of the process the bank needs only not respond to a request to halt the flow of data. Moreover, data cannot be transferred without the blockchain being checked for the relevant permission first.

Containers 5, 6, and 7 carry out the corresponding functions in the second bank, with the internal system being represented by 8. As can be seen, each bank has only one point of communication with third parties (containers 1 and 5, respectively), and white listed banks are connected to other another at two levels; the blockchain level via containers 2 and 6, and the API level through containers 3 and 7. In the event of a successful request from the second bank, where 4 has sent the data requested via API to 3. In turn, 3 will communicate the data to 7. The data is encrypted with the public key that the requesting bank used to sign its blockchain transaction. This is the same public key address that is associated with customer's data access permissions for the bank. As such, in addition to the data being sent via an encrypted channel the data is encrypted such that only the genuine recipient can access it. This ensures no data breaches can take place during transit.

### 5.3.7 The Smart Contracts

The system design includes two Smart Contracts. The access control smart contract records end-client permissions for banks to access their data. The second contract facilitates the

end-client's on-boarding at new banks without creating duplicate on-chain identities. In both cases, transactions are always sent by banks on behalf of their customers due to the chain being private, it is not visible to anyone apart from the banks themselves.

The access control contract is designed to record, by way of on-chain ID's for customers, banks, and files, which banks have access to what. The records can be created and updated, and removed, by sending transactions to the contract. When a request for data is made by a bank the system consults this contract to check that the appropriate permission exists, and has not been removed.

The on-boarding contract stores records of single use authentication pass-code, which customers present to a bank they wish to on-board. The pass-code allows the bank to know their on-chain ID and, thereby, access data to which they have been given permission. The customer asks their current bank to generate a single use authentication pass-code. The bank generates the pass-code, then generates a hash of it. The current bank sends the hash of the pass-code, along with the customer's on-chain ID, to the on-boarding contract. The on-boarding contract associates the pass-code hash and on-chain ID by storing them in a mapping. When the customer presents the pass-code to the new bank, it is able to prove the client wishes to do business with it by sending the pass-code to the smart contract via a transaction. This contract generates the hash of the pass-code sent by the new bank, and checks it against the list of currently existing and unused pass-code hashes. If there is a match the associated on-chain ID for the customer will be sent back to the bank. The bank can then use the customer ID to make requests for their new customer's data. The system as a whole only ever sees pass-code hashes and associated on-chain ID's. An observer gains no information about the identity of any of the actors involved. As the pass-codes are single use they cannot be reused to gain access to data. If the pass-code is ever lost or exposed the current bank can, upon receiving notification from the client, send a follow-up transaction to the contract declaring that token (and its hash) void, and issue a new pass-code to the client repeat to process of association. It also ensures that clients can retain one ID across

all banks. Functionality surrounding the on-boarding contract is a work in progress and so is not discussed in the implementation or results sections.

## 5.4 Implementation

This section provides details of how the design has been translated into a functioning implementation. In the current prototype the logic within each container, and the API implementations are written primarily in Node.js [37]. For this section the focus is on the Ethereum based implementation of the system. Scripts for first-time setup, used for configuring geth and launching the smart contracts, are written in Python3 [38].

### 5.4.1 Communications

As mentioned in section 5.3 all parts of the system are containerised; Docker is utilised for this. Each container has open only the ports it requires to operate. The containers run by each bank are part of the same Docker virtual network. This allows the system to take advantage of Docker's internal DNS resolution, with containers being referred to by name, rather than by IP. This removes the need to fix IP addresses for containers when communicating internally. In terms of security, running all the containers on a Docker network allows them to communicate while remaining isolated from external communications where necessary.

The API in use is an extension of the PSD2 compliant open banking API [85]. As this API is designed for the acquisition of customer information, it was necessary to extend it in order to provide some of specific inter-container communication. Each container in the system contains a subset of the full API, depending on the tasks it is required to carry out. The API is written in Node.js.

### 5.4.2 Smart Contract

The access control smart contract was written in Solidity and compiled using version 0.4.26 of the compiler via the online remix tool [27]. It makes use of mappings wherever possible to avoid potentially lengthened execution times causing by iterating over arrays. Arrays in Ethereum can give unpredictable execution time as they grow. Moreover removing elements leaves gaps in the array, requiring manual reordering. A number of data structures are used in order to store file hashes associated with an end-client and to store access permissions associated with files. Permissions take the form of white lists, if a bank's blockchain ID is not explicitly listed as having permission then the contract's checking function(s) will return a false value.

### 5.4.3 Blockchain Platform Container

The system makes use of a private, permissioned Ethereum chain with each bank running the geth Ethereum client software. Each bank has one verifier address, creating a 'one bank one vote' system. Ethereum's PoA implementation, Clique, is used as as the consensus mechanism. This allows the system to operate rapidly, the system has been configured to produce a block only when transactions are received. This means there are no empty blocks causing the chain to grow unnecessarily and that transactions are processed as rapidly as possible.

The details discussed above are defined in the genesis block of the chain. All peers must share the same genesis block in order to make an initial connection to the blockchain network. Puppeth, a CLI tool that comes bundled with geth, is used to configure the genesis block. The genesis block is then stored in each container and copied to the designated chain data folder on the host system as part of the first-time setup.

For the geth software the default listening port of 30303 is used to communicate with blockchain node peers externally, and port 80 is used to communicate with the network peers'

containers via API calls to and from the DAC container. Inside the container the Node.js software communicates with geth via the provided ICP interface on port 8545.

### 5.4.4 Data Storage and Anonymity

When documents are added to the system a hash is generated from the document and a single use, secret salt. Under the system salts are randomly generated strings. Each file has its own separate, unique salt. The SHA3 algorithm is used to generate strong cryptographic hashes. Each salt is stored by the bank in their internal storage system and associated with the relevant file. The hash is stored on-chain and associated with the end-client's random ID via mappings in the access control smart contract. Each hash is recorded under a given file type: passport, ID card, etc. If the customer gives permission to a specific bank to access a file then the white list will be updated via a transaction. When responding to a valid request for a file the DAC container will request the file and salt from the bank's internal system, encrypt them both and send both to the receiving bank. Encryption is done using the receiving bank's public key, which doubles as their on-chain ID for Ethereum. Doing so ensures that only the intended recipient can decrypt the files using the corresponding private key. After decrypting the received data, the bank can then generate the hash using the file and the salt, and check it against the hash stored on-chain to ensure data integrity. In the event of a valid data deletion request the bank must remove both the file and a the salt in order to make the on-chain hash anonymous to them.

### 5.4.5 The DAC Container and file Transfer

Next, a closer look at how file transfer is handled by the system. The DAC container is responsible for acting upon the result of on-chain access requests. The DAC container must receive a confirmation of validity from the blockchain platform (BP) container before fulfilling a request. The DAC encrypts the file and salt to be sent with the requesting bank's public

key, such that they can only be decrypted by the intended recipient. The encrypted data is made available by placing it in the FTP server which runs inside the DAC container. The data is held in memory and never written to the container's storage. A one-time URL is generated for the files, this is also encrypted with the requester's public key. This URL is then sent directly to the requester's DAC container via an API call. The receiving DAC will access the URL to retrieve the files. After decryption it will recalculate the hash and check it against the hash for the file it requested. Should the hash not match, a dispute can be logged on the blockchain via a separate transaction. In this case an API call is made to the complainant's local BP container, which will generate and send the appropriate transaction to the access control smart contract. Once the URL has been used to retrieve the files, the file sender removes the encrypted files from memory and the URL is considered spent, meaning it can no longer be accessed. This is done in order to protect the files by ensuring that files exist outside the bank's secure storage system for the minimum necessary time.

### 5.4.6 Deployment

Here, the two technical requirements for banks implementing the system, are discussed. It is required that the banks install Docker, such that they are able to run the containers. Banks must implement part of the API in order for the DAC container to be able to request files from the bank's internal system. This can be a significant technical undertaking for the banks as it requires the production of code on their part. However, as a predefined API is being implemented it is much simpler than attempting to build a custom interface between the DAC container and each bank's internal system. The API is implemented once, after which containers can be updated and changed, as can the bank's internal systems, without any undue disruption to the system on either side of the API.

**Table 5.1** Iterations and success rates of load tests

| Test | Throughput (Iterations/s) | | Average latency (s) | |
|---|---|---|---|---|
| | Cali. | Paris | Cali. | Paris |
| Add File | 1.51 | 1.52 | 5.68 | 6.38 |
| Grant File Access | 0.92 | 1.00 | 9.98 | 9.18 |
| Grant Tier Access | 0.86 | 0.81 | 10.25 | 11.09 |
| Change Tier Access | 0.66 | 0.69 | 10.67 | 13.01 |
| Remove File Access | 0.66 | 0.32 | 13.33 | 22.5 |

## 5.5 Experimental Results

This section details the results of load testing a 3 instance setup of the blockchain-based functionality of the system, namely the DAC BP containers. Tests run for several different scenarios, each of which consists of API requests sent from the host system to its locally run DAC container. These requests trigger communication between the local DAC and BP containers. In turn the BP containers of each instance communicate with each other to perform on-chain operations which propagate throughout the network. A network of 3 instances of the system was built, with each node running in disparate global locations: California, Paris, Tokyo. All instances are connected to each other and all run an image of the DAC and BP containers. A fully functional private blockchain, hosting the access control smart contract, is in use. All instances are PoA verifiers defined in the genesis block. Each instance is hosted on a t3.medium AWS server with 4GiB RAM and 2vCPU's with a 2.5GHz clock speed, all servers run Ubuntu 18.04.

5 different tests were run; they replicate the kind of operations that would be carried out by banks utilising the system. They are detailed in table 5.2 in order of the number of blockchain transactions each test requires. Each test is run for 100 seconds with requests

**Table 5.2** Load tests and their operations

| Test | Functions | Transactions | Local chain reads | HTTP requests |
|---|---|---|---|---|
| Add File | dd customer, add file | 2 | 0 | 2 |
| Grant File Access | add customer, add file, grant file access, check file access | 3 | 1 | 4 |
| Grant Tier Access | add customer, add file, grant tier access, check tier access | 3 | 1 | 4 |
| Change Tier Access | add customer, add file, grant tier access, check tier access, change tier access, check tier access | 4 | 2 | 6 |
| Remove File Access | add customer, add file, grant tier access, grant file access, check file access, remove file access, check file access | 5 | 2 | 7 |

sent in parallel; batches of 10 with 2 second intervals. Each request waits to receive the appropriate response from the DAC container, how long this takes was recorded as a measure of latency. Throughput is taken to be the number of requests the system fully processed during the 100 second window. The tests are run twice, once with the requests originating from the server located in California, and once with requests originating from the server located in Paris. Table 5.1 shows system's throughput and latency for each of the tests.

Tests run from both California and Paris show comparable throughput and latency, suggesting that the system is capable of on-boarding and/or performing complex on-chain operations for around 10 customers every 2 seconds. Moreover, the rules of the Remove File Access test show that on average the system is capable of the full cycle of adding a customer, granting and checking access, and then removing access and checking again between once every 1.6 second and once every 3.3 seconds. This test carries out these operations much faster than a customer is likely to in a real-world scenario, showing that the system is capable of handling even a large influx of customers with complex operational requirements.

As can be expected, local read-only operations on-chain data are much faster than full blockchain transactions. Geth automatically makes a local query for calls to any functions that do not affect state, as oppose to sending a full blockchain transaction for anything that will alter the state. In the case at hand, local calls are used in any 'check' function (check file access, for example). In the Paris Remove File Access test the average time between sending a HTTP request to check file access and receiving a response is 39ms, whereas the average for adding a file (a full blockchain transaction) is 3,698ms. Here one can see the difference between the time to achieve consensus between internationally located instances under a high transaction volume, and to access a local copy of the chain is 3,659ms. However, in both cases the time is small enough to ensure that the system is well within the realms of acceptability for both banks and customers. It is also likely to be the case that the response time for requests which include a blockchain transaction is a lot lower when the system is not under such a high load. Realistically speaking a deployed system is not likely to see the

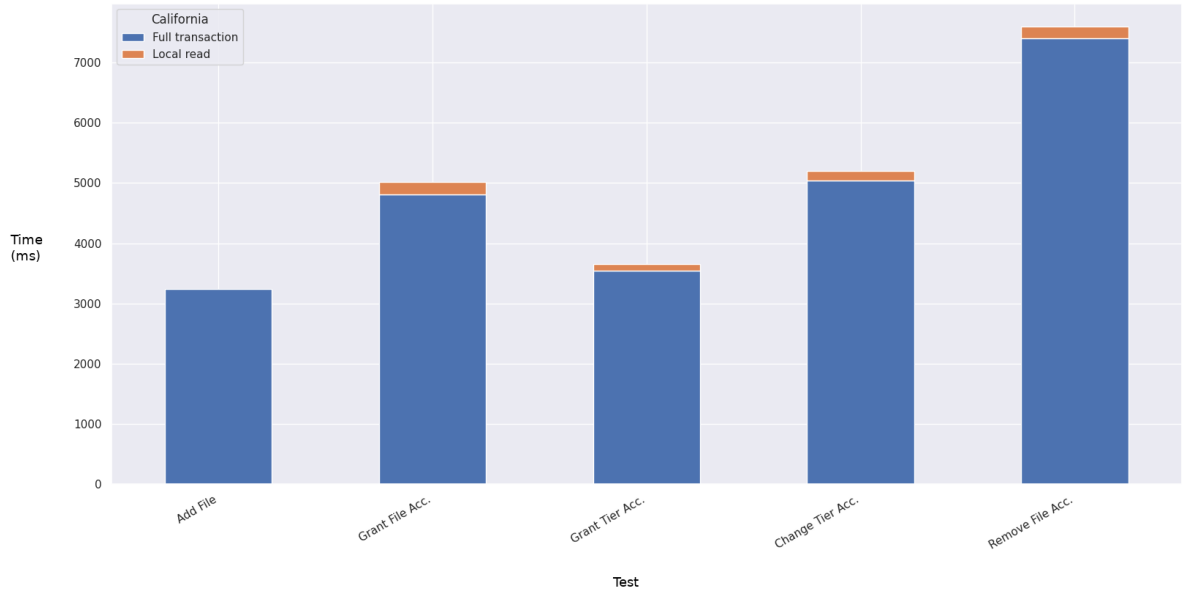


**Figure 5.4** Average chain interaction times for tests on California server

addition of 150 new customers every 100 seconds. The average times for each test, including average on and off-chain operation times can be see in figure 5.4 and figure 5.5 for California and Paris, respectively.

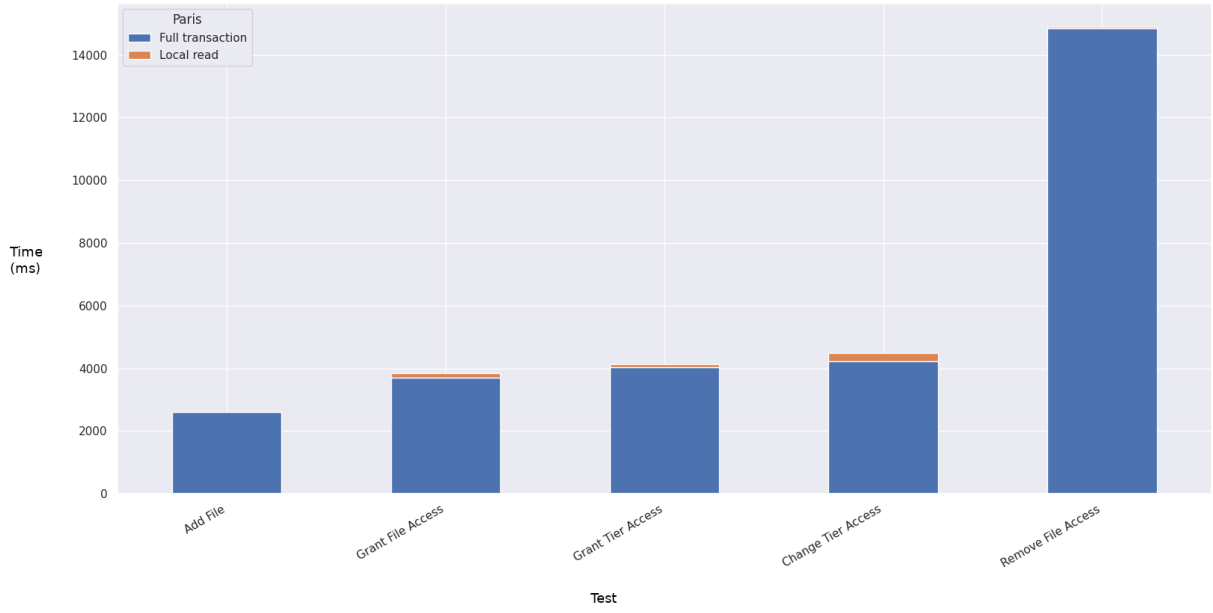


**Figure 5.5** Average chain interaction times for tests on Paris server

## 5.6 Conclusion

This chapter details a system designed to reduce the costs of KYC compliance while ensuring all data security and privacy requirements are met in order to provide a useful and attractive system. The development of the system is guided by the real-world needs of the industry partners. Having multiple industry partners allows for the proposal of a system capable of serving the sector at large. The system meets the needs of the industry partners, including reduction or KYC costs, and removal of over reliance on centralised third parties for storage or data acquisition, which in itself helps further reduce costs.

The system was load tested with 3 instances of the software located globally. Repeated calls are made to the system's API's in order to understand how well the software performs and whether the various components can interact with one another fast enough when the system is under a heavy operational load. The system is capable of adding 150 new customers every 100 seconds.

Future work includes running load tests with all security mechanisms in place (HTTPS, for example) as well as measuring document exchange times across disparate nodes.

# Chapter Six

# A New Kind of Marketplace: Pricing of Data

> "Money is a tool of exchange, which can't exist unless there are goods produced and men able to produce them."
>
> Ayn Rand

## 6.1 Introduction

The burden of compliance for KYC and AML legalisation has never been higher for banks operating within the EU. KYC refers to the legal obligations banks have to know who they are doing business with. AML legislation embodies an associated legal obligation by which banks must be sure that they are not aiding in the laundering of illegally obtained money. One of the banks' major pain points for compliance with KYC and AML laws is the collection and verification of documents. The processes tend to be manual, and are repeated by each bank for the same customer. The aim is to reduce the need for manual collection through inter-bank data trading.

The GDPR came into force in 2016 [101]; it also has compliance implications for banks.

It states that which entities have access to data must be controlled by the DO, with the DO being the person whom the data is about. Moreover, the DO must be informed of the intended use of their data, and must give explicit consent for its use. In addition, providing other legal requirements do not override GDPR (AML laws, for example), financial institutions must delete the DO's data at their request.

Given the burden of compliance, and the costs that it currently carries with it, there is an appetite in the financial sector for collaboration to reduce costs [3], especially when the likelihood to have multiple accounts across multiple institutions is considered [86]. One study finds that 50% of Americans hold accounts with at least two banks, with 22% holding accounts with two or more banks [45]. Blockchain is able to allow for decentralised collaboration amongst actors that do not fully trust one another without the need for a trusted third party, and as such has been recognised as being able to reduce costs in the financial sector [62]. In order for blockchain-based collaboration to be properly incentivised, an appropriate method to price data is required.

The previous chapter detailed a blockchain-based data sharing project which enables GDPR compliant KYC data sharing between banks. Customer permissions and records of access are recorded on-chain by a smart contract.

The work reported in this chapter augments the existing blockchain-based sharing system by providing a method to price data such that banks are able to recoup their costs, and that they are properly incentivised to participate in data trading. The goal is to provide a method of pricing data such that banks with different quantities of data, and different interests, can all benefit by reducing costs, and in some cases profiting from the data they collect.

Both the previous chapter and this work take their requirements directly from the three financial institutions with which we work. These industry partners vary in scope, and nature of business, ranging from national to international, and from retail to corporate banking. Each partner operates within the EU, and as such the focus is on compliant data trading for EU level laws. However, it is assumed that data can be traded across national borders. The

pricing requirements and incentives are derived from requirements gathering and iterative design. The proposed pricing method is shaped by, and designed to meet, industry needs.

The remainder of this chapter is organised as follows: Section 6.2 details the unique features of the marketplace the blockchain system creates, and the needs of different actors within the system. Section 6.3 details the pricing model designed to fit the KYC data trading use case, and section 6.5 provides the conclusion and future work.

## 6.2 The Marketplace and its Actors

Due to the requirement that data can only be exchanged with the customer's permission, the system does not operate a free market and has the following characteristics:

- **Permissions:** A customer's permission is required in order for the data to be exchanged between participating banks.
- **Purpose:** There can be no prospective purchasing, or trading for any purpose other than KYC compliance. The GDPR requires that DC's must make clear the purpose for which they intend to use the data. Moreover, the system does deal with permissions for any other kind of use.
- **Payment:** The system is built to reward the entity which incurred the cost of the original data collection (the data seller) while reducing costs for data purchasers. The original collector is inherently known through the blockchain.
- **Different Actors:** The deployed system needs to provide functionality and incentives for banks with different sizes, and as a result, different needs. In addition, the marketplace could be expanded to allow for the inclusion of entities other than banks, whether they are companies which can provide KYC data, or companies from other sectors which have similar KYC needs to the banks. The different kinds of actors are

summarised in table 6.1, the two categories that banks fall into can be seen as well as possible future actors (marked with a *).

- **Multiple Roles:** Each participating bank can be both a purchaser and a seller, although it is likely that some banks will be largely one or the other.

**Table 6.1** Different marketplace actors and their definitions

| Actor | Entity | Definition |
|---|---|---|
| Mostly seller | Large bank | Bank that has a large customer base |
| Mostly purchaser | Med/Small bank | Smaller banks or ones which are growing may find cost of full KYC prohibitive |
| Exclusively seller* | KYC service provider | Entity which collects/verifies documents on behalf of banks |
| Exclusively purchaser* | KYC service user | Non-bank entity that is required to carry out KYC, possibly law firms or Funds |

Figure 6.1 shows Bank A as the original collector of a customer's documents. Assuming the customer has granted permission to Banks B and C, Bank A is able to trade the documents with them, in return for on-chain tokens which hold real world value and are used to pay for data within the system, the token is discussed in detail in 6.3.2. It should be noted that the system offers no way for Banks B and C to trade the documents. As the bank that originally collected the documents is known due to the blockchain record, only they can be paid for the documents they collect and verify.

With regard to the data sharing system which this chapter builds upon, it should be noted that it is an Ethereum private-chain, with each participating bank operating one node and having one vote. Ethereum's PoA implementation, Clique, is used to provide fast and efficient consensus [26]. It makes use of a smart contract known as the permissions contract, which stores data access permissions that have been granted by the customer; this ensures

GDPR compliance. It is written in Solidity and provides a record of which banks have access to which customer documents. The focus of the system is to facilitate data sharing between banks and remove reliance on third parties in order to reduce costs.

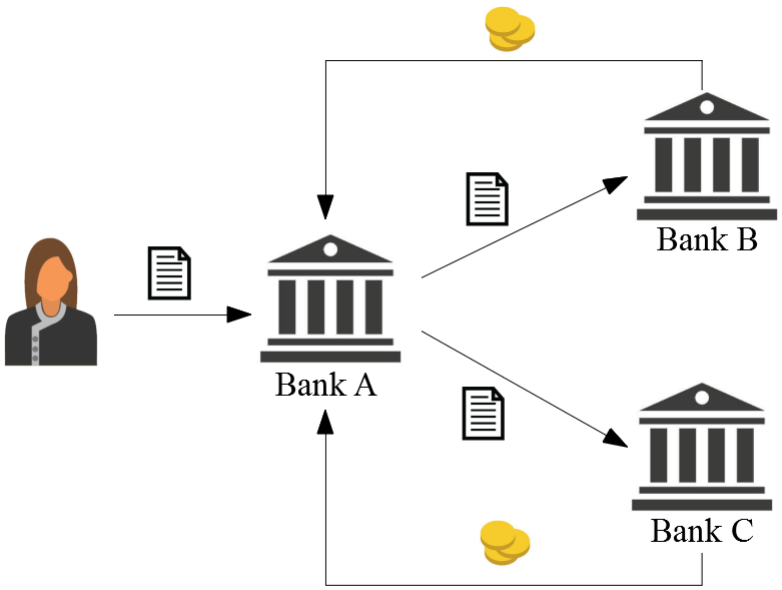


**Figure 6.1** Bank A collecting documents from the customer and selling them to Banks B and C

In the following section pricing methods are discussed, with the above requirements in mind. As discussed in section 3.5, pricing methods from the existing literature do not fit exactly into the unique marketplace that has been created.

Due to the large variety of documents that the system is designed to handle the concept of tiers was developed. Documents are divided into tiers based on the complexity and service for which they are required. Table 6.2 shows the first 4 tiers which deal with basic KYC document requirements for natural persons and companies. The tiers have been developed based on feedback from industry partners about the documents they collect, and the complexity pertaining to different types of customers. Examination is restricted to the first 4 tiers for the sake of simplicity, however it is a simple process to extent the system to allow for further categories. For example, documentary requirements could be categorised based

**Table 6.2** The first 4 tiers into which documents are organised

| Tier | Purpose | Definition | Documents |
|---|---|---|---|
| 1 | Open account/ basic KYC | Basic documents all natural persons must provide | ID card, Proof of address |
| 2 | Open account/ basic KYC | Documents for EU nationals | Job contract, Residence permit |
| 3 | Open account/ basic KYC | Documents for third country nationals | US tax documentation |
| 4 | Basic KYC of companies | Required for providing services to legal entities | Certificate of incorporation, Extract from business register, Proof of power of signature |

on the size of an organisation in higher tiers. It should be noted that the first 3 tiers are cumulative, if an individual requires tier 2 KYC verification they will also require tier 1.

## 6.3 Pricing

In this section the various aspects of pricing data for the specific blockchain-based marketplace at hand are detailed. While in section 6.3.1 the data pricing model is detailed, section 6.3.2 looks at the role of smart contracts to realise pricing within the data sharing system, and details the exchange between on-chain tokens and fiat currency.

### 6.3.1 Pricing Model

In order to mitigate the costs associated with KYC, it is necessary that data collectors are incentivised to share their data. This requires that data is priced for sale such that all parties

can benefit. As stated in section 6.2, the customer's permission is required to access the data in accordance with GDPR. As such, it is not a regular marketplace in which any purchaser is free to purchase the data if they wish.

Our pricing method leverages the fact that: i) Data is non-rivalrous, meaning that it being sold to one entity does not reduce the quantity, and it can be sold again. This is useful as it is often the case that customers will hold accounts with multiple banks. ii) The documentation required will be the same or very similar across banks; the same data can be sold to multiple banks. This enables us to price such that sellers profit while purchasers pay only a fraction of the total cost. Detailed below is a pricing model with this, and the observations from section 6.2 in mind:

The seller will charge purchasers price $p$, which is derived from the cost of document collection incurred by the seller $c$ and the number of times $n$ the original collector expects to sell the data (expected number of sales). Data collection is split into acquisition cost $a$ and verification cost $q$.

$$p = c/n,\ c = a + q \tag{6.1}$$

Splitting acquisition and verification costs allows for documents to be traded with or without the verification for that document also being traded, which is useful in the event that a bank is not allowed to share a check due to existing contractual obligations, or a purchasing bank wishes to carry out their own checks on a document instead.

In order to properly incentivise sellers, a percentage markup $x$ is included, and expressed as a value between 0 and 1 in equation (6.2).

$$p = c * (1 + x)/n \tag{6.2}$$

The inclusion of the markup enables sellers to not only recover their costs, but also profit from doing so. They can use the profit to offset the cost of data for which there are no

purchasers. Meanwhile, as purchasers are charged a fraction of the cost plus markup they are still saving on their data acquisition costs.

Assuming $n$ is accurate, equation (6.3) is used to calculate the $p$, including the special case of $n = 1$. In this case, the cost is split between the selling bank and the purchasing bank by calculating $p$ using $n = 2$, the purchasing bank is charged $p$, resulting in 50% cost reduction for the selling bank and a 45% cost reduction for the purchasing bank, with them paying a price equal to that when there are two purchasers.

$$p = \begin{cases} c * (1 + x)/n, & n > 1 \\ c * (1 + x)/2, & n = 1 \end{cases} \tag{6.3}$$

As the pricing algorithm relies on banks accurately predicting the number of banks a client will do business with, it is possible that a particular document will be bought more or less times than expected.

The underselling of documents (bought fewer times than expected) should be avoided as it impacts the revenue of the selling bank, as they need the number of expected sales $n$ to be met in order to receive the full amount of expected revenue. It is expected that banks will be able to use their existing data sets to estimate $n$ with a reasonable accuracy. Moreover, the predictions can be fine tuned over time as it is observed which documents for which types of customer did and did not meet the expected value of $n$.

It should be noted that it is possible to set expected sales low in order to increase the chances of the selling bank to acquiring its total revenue. In the long term this would not affect purchasing banks due to the redistribution method shown in equations (6.4) and (6.5). However, in the short term low values of $n$ could lead to increased costs for purchasing banks. Moreover, accurately predicted values of $n$ allow both purchasers and sellers to better forecast their expected costs and revenue.

In the event that data is oversold, meaning the number of actual sales $k$ has exceeded the expected sales $n$ ($k > n$), it is possible to redistribute the costs such that each purchasing

bank has paid the same. A method of redistribution detailed in Moyano et al. [71] is used as follows: The new purchasing bank pays $p'$ calculated using $k+1$.

$$p' = \begin{cases} c * (1+x)/(k+1), & 1 < n \leq k \\ p, & n = 0, 1 \text{ and } k = 1 \end{cases} \tag{6.4}$$

Our system of remuneration for the data seller requires that a special case is accounted for; in the event that $n = 0, 1$ and $k = 1$ the second purchasing bank will pay the price of the previous sale $p' = p$ which is sent to the selling bank. This results in the selling bank securing its total revenue, and the two purchasing banks paying equal amounts. Equation (6.4) is used to calculate $p'$.

Redistribution $r$ is calculated using equation (6.5). The value of $r$ is sent to each of the $k$ banks that have already bought the data. As this process only takes places when $k > n$ the selling bank is able to gain the full amount of expected revenue, while purchasing banks further reduce costs.

$$r = p'/k \tag{6.5}$$

Unlike previous work, this pricing method enables data sellers to profit, rather than just recover costs. Moreover, it reduces KYC costs for the banks, thanks to the high likelihood that data is sold multiple times, the selling price is much lower than the cost of acquisition even with the markup included. In addition, the ability to redistribute funds between purchasers is retained; this ensures an even distribution of costs.

Our pricing method has the advantages of being simple to implement as the equation can be coded into smart contract functionality. A smart contract is used to realise the pricing described above, as well as for escrow. The details are discussed in section 6.3.2.

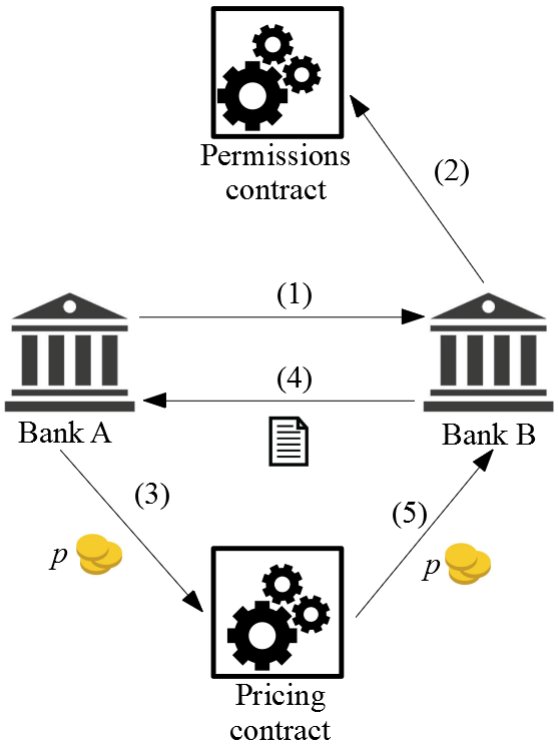


**Figure 6.2** Document purchasing process between two banks showing the roles of the Pricing contract

### 6.3.2 The Mechanics of Pricing

This section details the role of the pricing contract and the implementation of an on-chain token which can be used to pay for the traded data.

#### The Pricing Smart Contract

Building upon the existing data sharing system, an Ethereum smart contract, written in Solidity, has been implemented to record the agreed upon price of each document type, the markup, the number of expected sales based on the type of customer, and the number of actual sales. This makes pricing completely deterministic and gives every bank the assurance that their purchase and sell prices will be fair across the board.

The role of the smart contract is discussed below, figure 6.2 shows the the role the pricing contract plays in the process of banks trading data. Some steps are omitted for brevity:

1. Bank A requests a customer's data from Bank B.

2. Bank B checks the permissions contract to ensure that a record exists such that Bank A has access to the requested document(s).

3. Bank A queries the pricing contract for the cost of the document and then sends the appropriate amount in tokens to the contract. How the pricing contract handles the on-chain token used for trading is described in section 6.3.2.

4. Bank B sends the requested document to Bank A.

5. Once Bank B has confirmed receipt of the document the pricing contract releases the funds and sends them to Bank B.

The pricing contract holds a record of each purchase that has taken place, this is done in order to provide an audit trail and to allow for redistribution to take place as described below.

Once the expected number of sales $n$ has been reached the seller has gained their expected revenue through $n$ payments of the price $p$. Funds from purchases which exceed $n$ are redistributed by the smart contract by splitting the new price $p'$ equally between all banks that have already bought the data, thus decreasing the cost for purchasers. An example of the process is shown in figure 6.3 with the number of expected sales $n = 2$ $k = 3$, the steps are as follows:

1. Banks A and B pay the price $p$, which is calculated using the expected number of sales $n$. For each sale the pricing contract carries out the process shown in figure 6.2.

2. At a later point in time Bank C also purchases the data, as $n$ has been exceeded the contract charges Bank C the new price $p'$ which is calculated using equation(6.4).

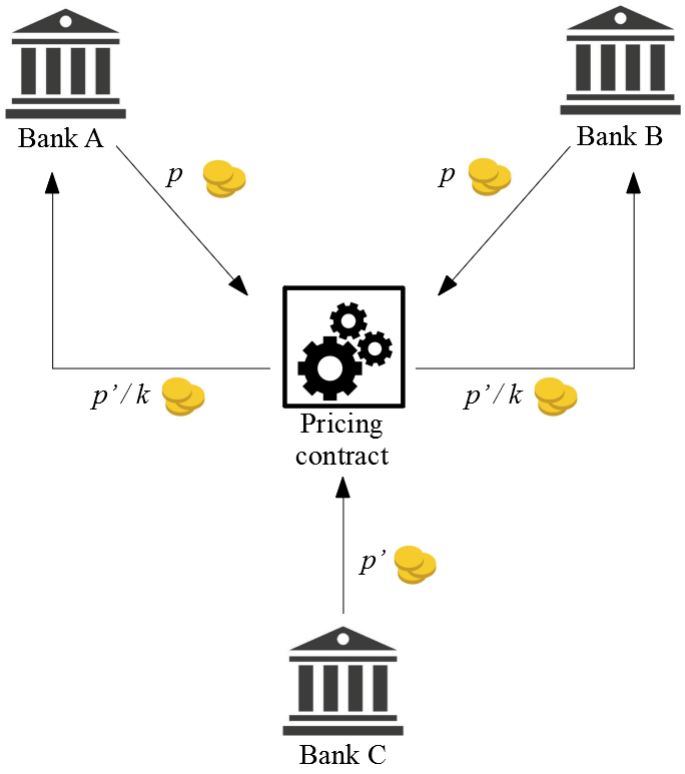


**Figure 6.3** Redistribution of data payment when $k = n = 2$ and the purchase $k+1$ takes place

3. Upon receiving the payment the contract divides $p'$ by the number of previous purchases $k$ and the resulting amount is sent to each of the previous $k$ purchasers of the data, in this case Banks A and B both receive $p'/2$.

The process is be repeated by the contract for any future purchases of the same data.

Lastly, it is necessary to consider how to set the values that the pricing contract stores: price, markup, and expected sales. These need to be agreed upon by participating banks, and will also need to change over time as it become necessary to update them, either to account for inflation, due to changes to the cost of acquisition, or due to changes in customer behaviour. This could be done either by a third party, or by voting amongst the banks. In accordance with the assumption of the rational actor, upon which blockchain rests, voting keeps the decentralised nature of the system, but could potentially leave the purchaser or the seller group worse off if one group out numbers the other and votes in their interest to

lower or raise prices, respectively. Conversely, there is no reliance on any third party to set prices, nor is any third party burdened with the task of setting prices.

### The Token

The concept of a token was briefly discussed in section 6.3. As the pricing contract handles the exchange of fund when a trade takes place, distribution of the token forms part of its functionality. This section goes into more detail as to how the tokens will be acquired and used, and how a banks can cash in and out of the system.

Within the system a token can be used to pay for data. Under Ethereum, tokens are fairly simple to define using a smart contract, and there are standards for doing so. Ethereum has various ERC's, one of which is ERC20: the Ethereum contract standard for a fungible token [103], its implementation as part of the pricing contract in this scenario allows for the pegged token to exist on the private-chain. As the token would have real world monetary value, the questions of how banks acquire it and exchange it arise.

Banks need to acquire tokens to begin trading within the system, and, as some banks are likely to purchase more than they sell, and some sell more than they purchase, there is a need to be able to exchange tokens both to and from fiat currency over time, as they will flow from purchaser to seller. This is achieved by assigning an entity to control the ERC20 based smart contract responsible for issuing the tokens to participants. This entity will be called the issuer, when a bank makes a payment to them, they are responsible for assigning the appropriate number of tokens to that bank's account within the ERC20 functionality of the pricing contract. As smart contracts cannot see, nor interact, with the external world, the issuer must be trusted to be honest in its issuances. The introduction of the issuer brings a third party, which must be used by the banks, into the system. While this breaks with the general theme of removing third parties, the issuer is not in a position where the banks would be reliant on them to the extent that the issuer could not be replaced.

As the issuer is trusted to honestly exchange fiat currency for on-chain tokens, their on-

chain behaviour is observable to all participants. In the event of a dishonest exchange, the value issued on-chain is trivial to verify, and the bank can resort to conventional payment system methods to prove the value of the fiat currency sent to the issuer. If such dishonest behaviour were to occur the issuer can be replaced by the banks which collectively control the system. On a technical level they can vote to remove the issuer from the chain. However, it is expected that there will be contractual obligations between the issuer and the banks, as there are with other entities with which banks do business, leading to off-chain consequences for dishonest behaviour. The replaceability and contractual obligations of the issuer render them unlikely to attempt to raise the cost of their services after taking on the role. Moreover, it increases the chances of them behaving honestly, as their choice becomes that of either generating revenue in an honest and agreeable way, or being removed from their role and generating no revenue.

It is possible to set the exchange rate in order to incentivise the issuer. For example if the Token/Euro rate was 1.001 then a bank would pay 1,000 EUR on a 1,000,000 EUR exchange, giving the issuer a gross profit of 1,000 EUR. The bank now owns 1,000,000 on-chain tokens and can use them to purchase KYC documents, while the issuer holds the 1,000,000 EUR in order to allow of exchanges back to fiat currency. Assuming this bank is primarily a purchaser, the tokens will slowly flow to the sellers the bank trades with until they need to purchase more.

The sellers on the other hand, will accumulate a surplus of tokens which they have no use for inside the system. They therefore have an interest in exchanging the tokens for fiat currency. This can be done by selling them back to the issuer. The issuer would take back the tokens and issue euros to the bank. Exchange rates or transaction fees can also be used here to incentivise the issuer.

Under such a system, purchasers can acquire the tokens to trade on-chain, sellers can turn their on-chain profits into more versatile fiat currency profit, and the issuer profits from providing their service as a neutral third party. Moreover, the issuer always holds the

required amount of fiat currency to allow banks to cash out of the system by exchanging their tokens.

## 6.4 Results

This section looks at the effect of the pricing method on the costs for different banks for different documents and customers. First, the assumed document and tier costs are introduced. Section 6.4.1 looks at the savings of purchasing and selling banks when trading an individual tier 1 document. Section 6.4.2 does the same for an individual customer based on tier cost. Section 6.4.3 analyses a scenario for the profits and savings of purchasing and selling banks might expect over the course of a year. Finally, section 6.4.4 gives an example of redistribution when the expected numbers of sales $n$ is exceeded.

Table 6.2 shows the assumed costs for individual documents and verifications for each tier, as well as the individual and cumulative costs of tiers. Tiers are used to calculate the cost of KYC processes for a given individual. As detailed in table 6.2, each customer falls into a given tier based on the type of customer they are. For the three tiers pertaining to natural persons it should be noted that to on-board a customer in a given tier the bank requires the documents from all tiers below the given tiers as well.

The assumptions made for the scenarios are given below. Sections 6.4.1 and 6.4.2 assume two purchasing banks. A 10% markup per document is assumed, unless otherwise stated. In contrast, in section 6.4.3 larger numbers of purchasing banks are considered. The expected number of sales $n$ is assumed to be accurate and data is not under or over sold, unless otherwise stated.

### 6.4.1 Pricing Individual Documents

Here an example is given for a customer which holds accounts with three banks who wishes to give their new ID card (a tier 1 document) to all three banks. An existing relationship

**Table 6.3** The individual and cumulative cost of the first 3 tier

| Tier | Number of Documents per Tier | Acquisition Cost per Document | Verification Cost per Document | Total Cost per Tier | Cumulative Cost per Tier |
|---|---|---|---|---|---|
| 1 | 2 | 30 | 30 | 120 | 120 |
| 2 | 2 | 30 | 30 | 120 | 240 |
| 3 | 1 | 45 | 45 | 90 | 330 |

with one bank and accounts being opened with two new banks over time ($n = 2$) is assumed. As such, equation (6.3) is used to calculate the price each of the two purchasing banks is charged. From equation (6.1) cost is made up of the acquisition of a tier one document is $a = 30$ EUR, and the cost of verifying it is $q = 30$ EUR (see table 6.3 for details of document costs). The total cost is $c = 30 + 30 = 60$ EUR. The customer needs to get the ID card to all three banks, and the bank that acquires and checks the document has a 10% markup $x$ therefore, according to equation (6.3), the price for each purchasing bank is $p = 60 * (1 + 0.1)/2 = 33$ EUR.

The bank that did the work (the selling bank) receives payment from the $n$ purchasing banks, meaning the seller earns $p*2 = 66$ EUR, giving them 6 EUR profit over the cost $c = 60$ EUR. The purchasing banks each save 45% (27 EUR) over the cost of manual acquisition and verification.

If each of the two purchasing banks wishes to carry out their own checks then acquisition cost $a = 30$ and verification cost $q = 0$ meaning $c = 30$ and each purchasing bank will be charged price $p = 30 * (1 + 0.1)/2 = 16.50$ EUR. In this case the selling bank earns $p * 2 = 33$ EUR, giving them 3 EUR profit. Each purchasing bank saves 13.5 EUR over the cost of manual collection.

### 6.4.2 Pricing a Customer's Data

Here an example of an individual requiring tier 3 KYC processes, namely a US citizen. In order to open an account for them, the bank requires all the KYC documents in the first three tiers. In this case $c_1$, $c_2$, $c_3$ represent the cost of the respective three tiers: $c_1 = (30+30)*2$, $c_2 = (30+30)*2$, $c_3 = 45+45$, and $c_t$ represents the cumulative cost of the required tiers: $c_t = c_1 + c_2 + c_3 = 330$ (see table 6.3). Using equation 6.3 with a cost $c = c_t$ this gives a price that the two purchasing banks will be charged to on-board a US citizen $p = 330 * (1 + 0.1)/2 = 181.50$ EUR.

The bank that collected and checked the documents (the selling bank) earns a 33 EUR profit instead of the $c = 330$ EUR expense incurred when doing full manual collection and verification outside the system. The two purchasing banks each reduced their costs by 45% (148.5 EUR) when compared to the cost of manual collection and verification.

### 6.4.3 Costs and Savings at Scale

In this section an example of the potential savings and profit banks could expect on an annual basis is given. A scenario in which a bank on-boards 10,000 new clients in a year who are EU nationals requiring tier 2 KYC processes is assumed. The bank carried out document collection and verification for all 10,000 customers. The costs per tier shown in table 6.3 show that a customer requiring tier 2 KYC documentation and verification will have a cumulative cost of 240 EUR, giving a total manual on-boarding cost of 2,400,000 EUR for 10,000 new clients.

Following the findings in [45] it is assumed that 50% of customers will hold accounts with two or more banks, this is the portion of customers whose data is salable, the break down is as follows: 28% have accounts with two banks, 11% with three, 4% with four and 7% with five or more (see table 6.4). This means that 50% (5,000) of the bank's new customers data will be bought by other banks.

**Table 6.4** The Revenue of selling bank and savings for each purchasing bank for customers with increasing numbers of data purchases

| Number of Purchases ($n$) | Percentage of clients | Selling Bank Revenue | Purchasing Bank Saving | Manual Acquisition Cost |
|---|---|---|---|---|
| 1 | 28 | 369,600 | 302,400 | 672,000 |
| 2 | 11 | 290,400 | 118,800 | 264,000 |
| 3 | 4 | 105,600 | 60,800 | 96,000 |
| 4 | 7 | 184,800 | 121,800 | 168,000 |
| | **Totals:** | **950,400** | **603,800** | **1,200,000** |

Assuming the $n$ expected sales is accurate, the price $p$ for each value of $n$ is calculated using equation (6.3). The selling bank's revenue is defined as $p * n * b$ where $b$ is the number of customers whose data is bought $n$ times. Each purchasing banks' total saving per value of $n$ is calculated as $(c * b) - (p * b)$ where $c$ is the cost of manual acquisition and verification.

Table 6.4 shows the amount the selling bank takes in revenue from its sales, and the amount that each purchasing bank saves compared to the cost of manual acquisition for the customers' data. The results show that the selling bank almost eliminates the KYC costs associated with the customers in question, while the purchasing bank reduces its costs by a little over half (50.32%). When the entire 10,000 customers, including those which are expected to hold an account with only one bank, are taken into consideration, the selling bank saves 39.6%, and each purchasing banks saves 25.16% compared to the cost of full manual acquisition and verification.

If the per file markup $x$ is increased from 10% to 40%, the selling bank generates a 9,600 EUR profit over the cost of acquisition of salable documents; the total saved for each purchasing bank is reduced to 441,200 EUR which is a 35.77% reduction on the cost of 5,000 tier 2 manual collections, as shown in table 6.4.

### 6.4.4 Redistribution

This section looks at how redistribution can increase purchasing banks' savings. As above, the example of an EU citizen requiring tier 2 KYC processes is used, it is assumed that banks purchase both the document and the verification, and that the expected number of sales has been determined to be $n = 2$.

When the number of actual sales $k$ is less than or equal to $n$, the purchaser's price $p = 132$ is calculated using equation (6.3). However if $k > n$, then for each sale beyond the value of $n$ a new price $p'$ is calculated using equation (6.4). The value of $p'$ is used to calculate the value of redistribution $r$ using equation (6.5), $r$ is sent to each of the $k$ previous purchasers of the data in question.

**Table 6.5** Purchaser price and saving changes with increasing values of $k$

| Number of sales $k$ | New price $p'$ | Redistribution $r$ | Saving over Manual Cost |
|---|---|---|---|
| 3 | 88 | 44 | 152 |
| 4 | 66 | 22 | 174 |
| 5 | 52.80 | 13.20 | 187.20 |

Table 6.5 shows the price $p'$ paid by each new purchaser, the value of $r$, and the savings over manual acquisition and verification for increasing values of $k > n$. It should be noted that once $n$ purchases have been made, the seller has received their full amount of revenue, meaning all purchasers' costs are reduced through redistribution for every purchase beyond $n$.

## 6.5 Conclusion and future work

This chapter proposes a method of pricing designed to function in a permission driven, blockchain-based, data sharing system. Unlike the state of the art it incentivises data holders

to share their data by allowing them to generate a profit from doing so, while still ensuring significant savings for the purchaser. It has been shown that a marketplace is not only possible, but highly beneficial for both purchasers and sellers in terms of cost reduction, even under the permission based restrictions of our GDPR compliant system. Moreover, the financial transactions take place without over reliance on any centralised third party to process the payments, this further reduces potential costs. Examples are given of how it is possible for banks to save at least 45% of their on-boarding costs for natural persons, while the bank providing the data can not recoup the costs of data collection and verification and make a 10% profit through permission based trading of the data.

In short, on-chain data pricing and payments are both possible and beneficial, even within the unique marketplace at hand, as is the exchange between fiat currency and on-chain tokens. Future work includes testing the system in a real world setting with our industry partners in order to fine tune the algorithms detailed in this work, and strengthening the system based on the observations and feedback this will provide.

# Chapter Seven

# Know Your Smart Contract

> "Along with the standard computer warranty agreement which said that [...] the purchaser should consider himself lucky to be allowed to give his money to the manufacturer, and that any attempt to treat what had just been paid for as the purchaser's own property would result in the attentions of serious men with menacing briefcases and very thin watches."
>
> Good Omens

## 7.1 Introduction

The contents of this chapter are drawn from the author's paper on the subject [79]. In turn, the paper is based upon the author's prior paper on the same subject [76].

The main contribution of this thesis is the KYC data sharing system. However, there is a related area of compliance: in the financial world the concept of Know Your Transaction already exists. It generally falls under the umbrella of AML law and refers to the requirement that entities which handle transactions must know their nature, including who they are from,

and who they are to. For example, extra checks are required for amounts over 10,000 euros. This requires not only that entities handling transactions know the amount being transacted, but that they are able to acquire further information when the rules require it.

In this chapter the concept of 'Know Your X' is extended to smart contracts. It is likely that in the future the use of smart contracts will become more widespread in the financial world. As such, it is envisioned that Know Your Smart Contract will play a role. In order to utilise smart contracts financial institutions should know what the contract does. With this in mind, a method of extracting defining features from contracts is detailed. Using a dataset as ground truth, it is possible to assign a purpose to each of these features. Such methods could be used by financial institutions to help provide assurance that they know the purpose and functionality of a contract with which they interact.

The application of the method also allows for the identification of areas that would benefit from standardisation. The ability to do so could help streamline ascertaining the functionality of contracts for financial institutions in the future, as standards provide set function names and interfaces for particular purposes.

Ethereum is the world's most popular blockchain for smart contract utilisation. It can be thought of as a blockchain-based, decentralised computer, with smart contracts as the programs it is capable of executing. Ethereum smart contracts are widely used and can hold a high value in cryptocurrency. The results of smart contract execution are agreed upon by consensus. In short, users can trust that the program was executed properly, in accordance with its compiled code, without trusting any of the individual nodes carrying out the execution.

In order to ensure all nodes to be capable of executing any given smart contract, the compiled code (bytecode) is permanently stored on the blockchain. When a user sends a transaction to a smart contract, the program's bytecode is executed with the parameters passed to it by the user that generated transaction. A transaction specifies the contract being called, which function to call, and the parameters to be passed to that function.

Ethereum bytecode consists of opcodes and static values. The Ethereum Virtual Machine (EVM) reads and operates on these opcodes and values. Control flow is dictated by the jump instruction (`JUMP`) and conditional jump instruction (`JUMPI`). Contracts are most commonly written in Solidity, a JavaScript-like language specifically for Ethereum smart contracts [23]. It compiles to Ethereum bytecode.

While bytecode is always available, the availability of smart contract source code is entirely at the discretion of the developer, and is unavailable in the majority of cases. At the time of writing [22] has verified source code for only 49,570 of the 46,338,837 contracts on the public blockchain. Etherscan provides a well known platform for viewing available source code. It is also possible for developers to release their source code through other channels, `github.com` for example, but this is likely to be less easily accessible to the end user.

Ethereum has a growing number of EIP. ERC's are a subset of EIP's, most of which define templates for contracts, for a given purpose. The most popular is currently ERC20 [103]. It provides a standardised set of functions that must be implemented for tokens in Ethereum. ERC's only detail the functions that must be implemented, not the semantics of implementation. In keeping with this approach, function names, and not function content are considered for defining contract purpose.

This chapter aims to provide insight into the purpose of contracts even when the source code is unavailable. Standardisation can be a lengthy process, and is often a reaction to an existing need. The method detailed aims to provide a method to speed up the process, by early identification of potential candidates for new ERC's. The approach can be used to identify where the efforts for standardisation could be focused.

In Ethereum smart contracts, function selectors are 4 byte identifiers created from the keccak256 hash of the function name and parameter types. They are stored in a contract's bytecode as static values and used to identify which function is being called. Selectors are utilised by treating those derived from functions in ERC's and those present in contracts, as sets which define contracts. The sets of selectors are used to cluster similar contracts.

A reverse look-up is used to label clusters. Selectors are guaranteed to be unique within a contract and are universally uniform. A contract implementing an ERC must have the selectors of that ERC.

In short, all contracts implementing a given ERC will have the same function selectors. To validate the results of the clustering, the selectors generated from the functions defined in ERC20 are used. Being able to cluster all ERC20 contracts together shows that the clustering method is capable of grouping contracts accurately.

## 7.2 Related Work

Ethereum's yellow paper is frequently updated. It clearly defines the opcodes and their behaviour [106]. Such information is necessary when working with bytecode.

Etherscan is a web-based blockchain explorer for Ethereum [22]. It verifies contracts by compiling the source code provided by developers and checking that the bytecode matches that which is stored on the blockchain. Etherscan can label smart contracts as ERC20 tokens without the source code. It is speculated that they may identify these contracts using features common to all ERC20 contracts, such as the function selectors.

Various projects and papers have focused on Ethereum bytecode. They fall into two main categories, those dealing with the monetary cost of executing smart contracts, and those dealing with contract security. Those dealing with cost include: [16], [15], [42], [66], [69]. The security focused projects include: [99], [17], [102].

In this work, use is made of the function selectors found in complied smart contracts. The website found at [2] provides access to a database of function selectors. It is populated with user provided functions which match a given selector. The entire database is available at [28], it is used to perform the reverse look-up from selectors to function names. The acquired function names are used to define cluster purpose.

Lastly, in our previous work we used clustering to identify contract purpose [75]. The

whole bytecode of each contract was used. We employed identification methods used in malware detection and used Etherscan's verified contracts to label clusters. In this work we improve the accuracy by focusing on the selectors.

## 7.3 Dataset

The raw data consists of the 1,499,926 contracts from Ethereum's public chain, between blocks 46,402 and 5,609,706. The function selectors are extracted from each contract and treated as elements of a set which represents the contract. Duplicates and contracts without selectors are removed. Duplicates are defined as contracts sharing the exact same set of selectors. If $A$ and $C$ are sets of selectors defining contracts then $C$ is a duplicate if $A = C$.

Timestamps in the raw data range from Aug-07-2015 04:42:15 AM +UTC to May-14-2018 01:52:40 AM +UTC; it contains 125,611 contracts which implement ERC's. 124,569 of these have an ERC set as a proper subset, and the remaining 1,042 as subsets. There are 68 ERC's listed at [29], of which the dataset contains 3. In order of prevalence they are: ERC20, ERC173, ERC721. ERC's 20 and 721 have both required and optional functions in their definitions; they are represented as two sets. One set, contains the required set of selectors that a contract must have in order to implement the ERC. A second set contains the optional selectors, that a contract may contain zero or more of. ERC20 and ERC721 are finalised, while ERC173 is still a draft.

With duplicates removed there are a total of 22,800 unique contracts, which are used for clustering. The number of duplicates each contract has is recorded and used as weight when clustering.

## 7.4 Methodology

This section discusses the clustering methods employed and the steps taken to build the dataset.

### 7.4.1 Validation

The validation method and the generation of function selectors is looked at first. Validation is carried out using the function selectors for ERC20. Selectors are split into the required and optional sets, with a contract being said to contain ERC20 if it contains all of the selectors in the required set. If ERC20 is a subset of a given contract: Let $A$ be the set of selectors of required functions of ERC20 and let $B$ be the set of selectors of optional functions for ERC20. Let $C$ be the selectors extracted from a given smart contract. $A \cup B$ defines the set of all selectors in the ERC and $C$ is said to implement the ERC if $A \cap C = A$

### 7.4.2 Selector Generation

In Ethereum, function selectors are used to identify which function is being called, and are a necessary part of contract bytecode. They are generated from the canonical form of the function, which is the function name followed by the type of each parameter, separated by commas, with no white space. For example, for ERC20:

```
function transfer(address _to, uint256 _value) public returns (bool success)
```

has the canonical form: `transfer(address,uint)`

The canonical form must be hashed using the keccak256 function. An interested reader might find it worth noting that, as Ethereum was up and running before the Keccak standard was finalised, the way padding is handled in Ethereum's implementation of keccak256 is different from the standard version of the function. The selector is taken to be the first 4 bytes of the hash, encoded as a hexadecimal number. The functions selectors for ERC's are generated

in order to check for their presence in the dataset. As such, one should take care to ensure hashing is done with the implementation specific to Ethereum. The input for the hash includes the data types of parameters, allowing for overloading. This ensures that functions with the same name and different parameters will have unique selectors. As the process produces uniform output, selectors will be the same for functions with the same name and parameters across contracts. This makes selectors a natural fit for representation as elements of sets. Set elements must be unique, a property each contract already has. As such, each ERC and contract can be represented by a set of its selectors. It is possible to validate the clustering through the use of set operations. Any contract implementing an ERC must contain the selectors of that ERC. In set theory terms a contract selector set $A$ must be a superset, of an ERC selector set $C$. If $A$ implements $C$ then: $A \supseteq C$

### 7.4.3 Selector Extraction

The dataset consists of a set of selectors per unique contract in the raw data. Selectors are extracted from the bytecode using pattern matching. The bytecode is converted from its hex form to human readable opcodes, which are required to generate selectors. The bytecode is shortened by discarding all code after the first instance for the `JUMPDEST` opcode. This is possible as contracts list all their selectors before the first jump destination.

```
PUSH4 0xdd62ed3e EQ PUSH2 0x0116 JUMPI DUP1
```

Selectors are identified using regular expressions to match the bytecode around each selector. An example using the selector of the first ERC20 function can be seen above, the selector can be seen after the `PUSH4` opcode. This is the code which performs the check to see if the function in question was called, and jumps to the location defined by the value after `PUSH2` if it was called. The extracted selectors are stored as one set per contract. The generic pattern within which selectors can be found takes the following form:

```
PUSH4 <selector> DUP2 EQ PUSH2 <value> JUMPI
```

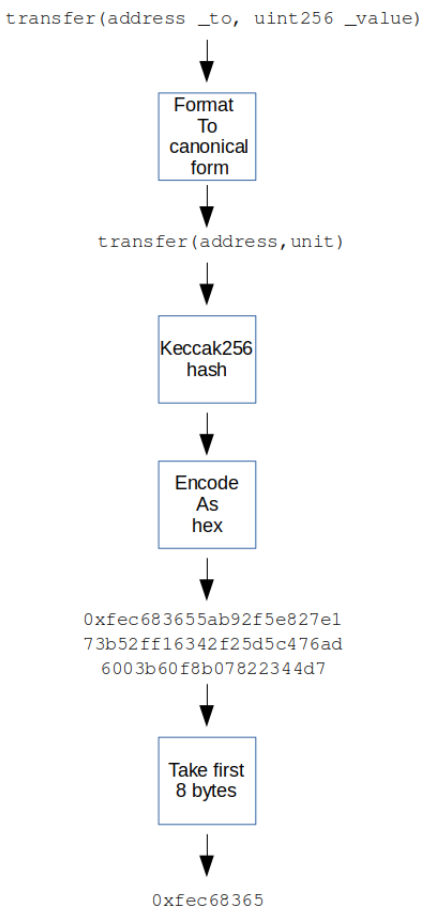


**Figure 7.1** Processing steps to generate Ethereum function selectors

with `<selector>` being the 4 byte from the keccak hash, which make up the selector. `DUP2` does not appear for all selectors. `EQ` checks if the function has been called, `PUSH2` and its proceeding value define the jump destination for the function and `JUMPI` moves the program pointer to the location at the given value if the result pushed to the stack by `EQ` is true.

### 7.4.4 Clustering

The K-means clustering algorithm is used to cluster contracts based on the selectors present in their bytecode. A sparse matrix with selectors as features, and unique contracts defined by the features they possess, is passed to the sklearn K-means implementation [19]. The contracts are weighted according to the number of duplicates they have. The weights are realised as a list with each element corresponding to a contract in the sparse matrix. Each element contains the number of occurrences of the corresponding contract.

K-means is run with different values of k: 2, 5 and 9. ERC20 is used to validate the results. The ERC20 contracts should be clustered together. The percentage of each cluster which is made up of ERC20 contracts is recorded. For each cluster the top 10 most commonly occurring selectors are extracted, and a reverse look-up from selectors to functions names is carried out using the database available from the 4byte website [28]. A more recent version of the database is provided via an API at [2]. The top selectors are used to provide suggestions for new ERC's.

Single Value Decompression (SVD), as implemented in the Python's sklearn library [19], is used to visualise the result of the clustering process.

## 7.5 Experimental Results

This section discusses the results of the clustering, and label clusters according to their member's purpose. Purpose is derived from the function names in each cluster. Starting with $k = 2$, the results are detailed. Increasing the value of k allows for further recommendations

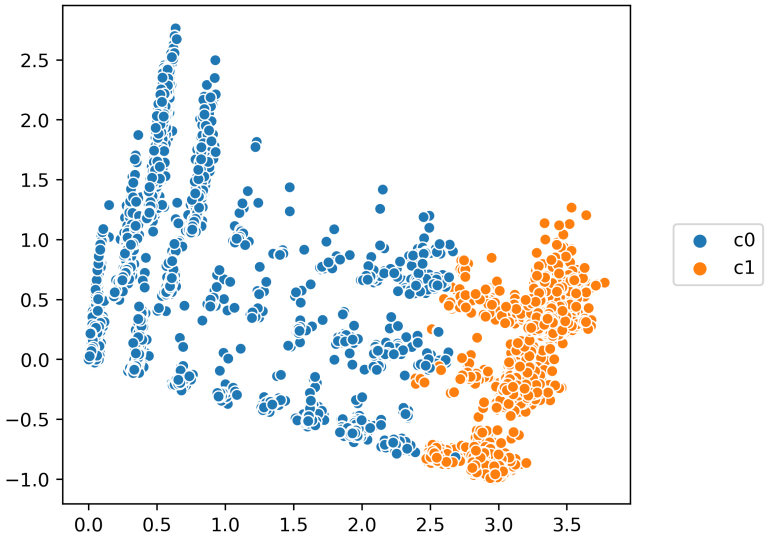


**Figure 7.2** Visualisation for clustering for $k = 2$

to be made. When discussing the contracts in a cluster, it should be understood that reference is being made to the unique contracts that make up the dataset, unless stated otherwise.

### 7.5.1 Clustering Results for $k = 2$

For $k = 2$, c0 represents non-ERC20 contracts and c1 contains the ERC20 contracts. c0 contains 16471 contacts and c1 contains 5817. The percentage of ERC20 contracts in each cluster is 1.3 and 93.3, respectively. With $k = 2$ is it possible to identify ERC20 contracts with a high level of accuracy.

### 7.5.2 Clustering Results for $k = 5$

By increasing the value of k to 5 it becomes possible to identify candidates for new ERC's. Fig. 7.3 shows the new clusters. Notably, the cluster on the left hand side has split in two. c4 contains the ERC20 contracts. It is larger than its equivalent for $k = 2$, containing 7490

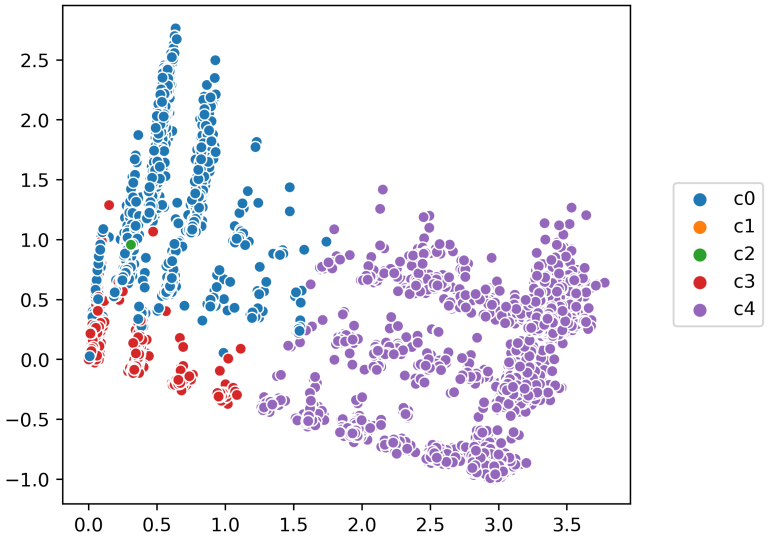


**Figure 7.3** Visualisation of clustering for $k = 5$

unique contracts, 75.4% of which are full ERC20 implementations. It contains a number of partial ERC20 implementations, with 7399 contracts containing the required ERC20 function `balanceOf()` and 6174 containing the required function `allowance(address,address)` (1225 less). The optional ERC20 function `symbol()` appears 7209 times, 1035 times more than the required allowance function. This suggests a large number of different ERC20 hybrids have been implemented.

Cluster c0 contains contracts which have functionality related to ownership: 5831/6330 contain `owner()` and 3444/6330 contain `transferOwnership(address)`. These two functions comprise the required functions defined in ERC173, suggesting that it is possible to identify the usage of this ERC. It appears that a great many contracts implement half of ERC173, by defining only the owner function. In c4; after the ERC20 functions, the most prevalent selectors are `owner()` and `transferOwnership(address)` which occur in 58.3% and 42.3% of the contracts respectively.

c1 contains one unique contract with a weight of 363285. This makes it a very commonly occurring contract. It contains all the ERC20 required and optional selectors. A

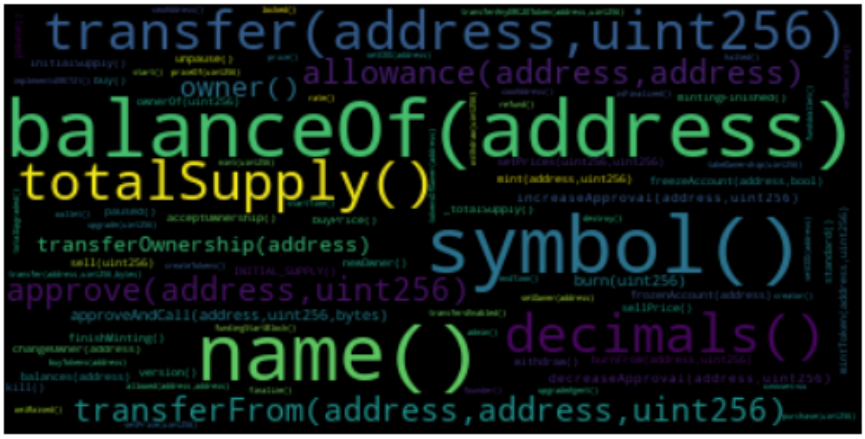


**Figure 7.4** Word cloud of functions in c4 for $k = 5$


**Figure 7.5** Word cloud of functions in c2 for $k = 5$

number of its functions are not found in the 4bytes database. After removing all ERC20 and unknown selectors, the selectors are as follows: `grant(address,uint256)`, `owner()` and `transferOwnership(address)`. The presence of ERC173 in such a heavily weighted cluster shows it is in frequent use. The presence of `grant(address,uint256)` shows a very commonly occurring function not present in any ERC. ERC1207, has a grant function, however it takes an extra string as a parameter.As ERC1207's grant deals with access control, it may be a popular access control function that should be standardised. Alternatively, ERC1207 is a draft at the time of writing, it could be modified to reflect the real world usage of grant functions.

c2 provides the first candidate for an ERC. It contains 16 unique contracts, and represents 56 contracts in the raw data. All the contracts contain:

`token()`, `bought_tokens()`, `sale()`, `set_token_address()`.

All but one contain:

`set_sale_address(address)`, `change_min_amount(uint256)`, `set_percent_reduction(uint256)`, `change_max_amount(uint256)`, `change_owner(address)`.

ERC900 has `token()` as one of its required functions, but none of its other functions are present. This cluster is a candidate for a new ERC. The function names suggest contracts used for crowd sales. As they do not contain any of the ERC20 selectors it is likely there is unstandardised, but desired, functionality. Analysis suggests a crowd sale ERC with required functions defined as those listed above.

c3 contains 8451 unique contracts. The most commonly occurring function is `kill()`, which occurs in only 1191 of the contracts. This cluster seems to catch contracts that do not display a strong similarity any others.

### 7.5.3 Clustering Results for $k = 9$

Finally, the results for $k = 9$ are detailed below, the clusters for which can be seen in fig 7.6. At a glance, boundaries between the major clusters have not vastly changed. However, a detailed look at the contents of the clusters allows more interesting information to be drawn out.

As before, c4 identifies the ERC20 contracts; 92.5% of the cluster is made up of contracts implementing all the required ERC20 functions. The hybrid implementations of ERC20 have been separated out into c1, with this cluster having a high presence of the required ERC20 functions; `balanceOf(address)` appears in 93.6% of the unique contracts, but full ERC implementations making up only 5.3% of the cluster. This shows that the clustering method is capable of distinguishing between full ERC20 implementations and partial ones.

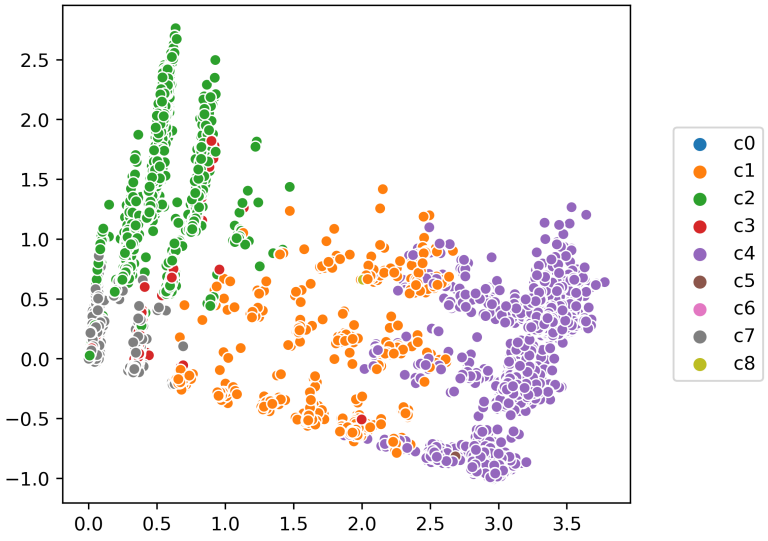


**Figure 7.6** Visualisation of clustering for $k = 9$

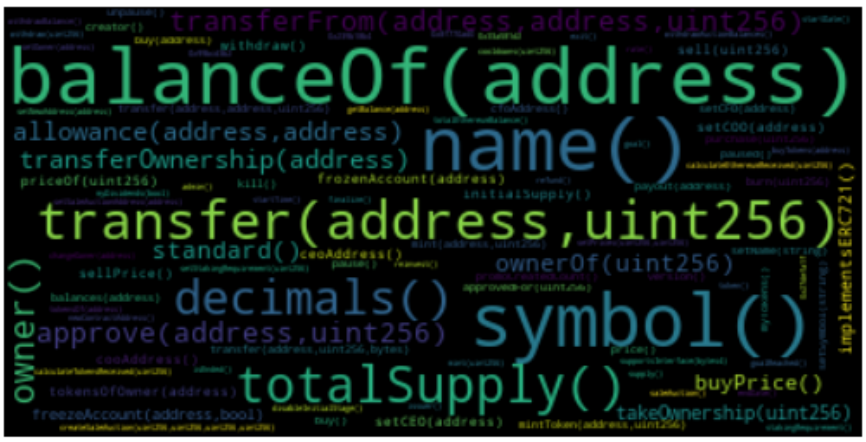


**Figure 7.7** Word cloud of functions in c1 for $k = 9$

It also suggests that people are either incorrectly implementing ERC20 or that they are using it as inspiration for building their own contracts. In c1 optional functions have a greater presence than some of the required functions, with `symbol()` appearing more frequently than some required functions. This shows the same trend for hybrid ERC20 implementations.: `transfer(address,uint256)`, `transferFrom(address,address,uint256)`, `approve(address,uint256)`, `allowance(address,address)`. Interestingly, the approve function is present in only 19.3% of contracts in the cluster.

c6 identifies the same heavily weighted, unique ERC20 contract as c1 for $k = 5$. The c7 cluster is extremely similar to c3 for $k = 5$. The number of contracts is almost identical and they display the same level of heterogeneity. As such, this cluster is still catching contracts which do not fit elsewhere.

c2, like c0 for $k = 5$, identifies contracts whose shared attributes are the functions defined in ERC173, with 95.9% containing the owner function. The trend for partial implementations of ERC173 is also visible here with only 55% of the contracts having implemented the required `transferOwnership(address)` function. The next most popular function, `token()` appears in 24.2% of the unique contracts. This suggests that people working with tokens tend to care about ownership, but the quantity is too low to suggest any other unifying purpose for this cluster.

The remaining clusters are c0, c3, c5 and c8. c0, c5 and c8 all display a high prevalence of functions not seen in the other clusters. They are analysed below.

c0 contains 7 unique contracts, with a combined weight of 140. All the contracts contain:
`signers, (uint256), activateSafeMode(), isSigner(address), safeMode()`
All but one contain:
`createForwarder(), getNextSequenceId(), sendMultiSig(address,uint256,`
`bytes,uint256,uint256,bytes)` None of these functions appear in any of the current ERC's. The names suggest an agreement schema. Although it is not entirely clear what the purpose of the agreement is from the function names, it may be possible to build a generic signature

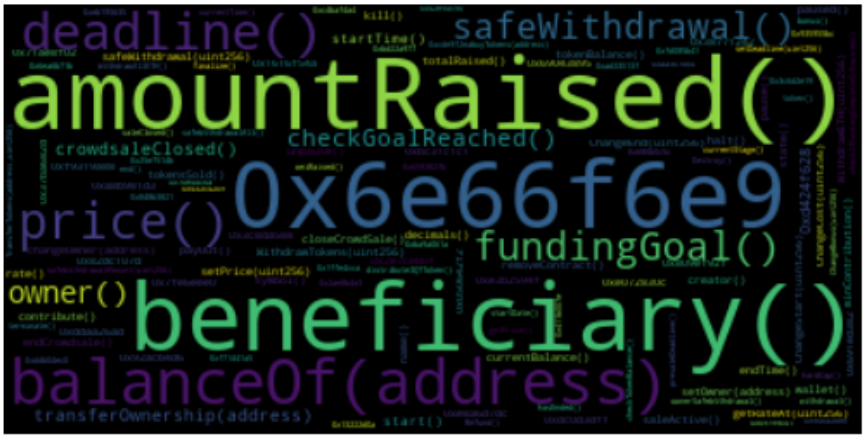


**Figure 7.8** Word cloud of functions in c3 for $k = 9$

and multi-signature ERC for on-chain agreement based on the above functions. The second ERC recommendation is put forward based on the above functions.

c3 has the following functions in more than 70% of its contracts: `amountRaised()`, `0x6e66f6e9`, `beneficiary()`, `balanceOf(address)`. With the exception of the selector for which no name is available, these functions suggest contracts for fund raising. Given the number of functions pertaining to funding, it is posited that a candidate for an ERC is a template for the kind of fund raising suggested by the functions in this cluster. An extension of ERC20 could include these functions to allow users to check how much has been raised and to whom ether or tokens are being given.

c5 consists of 45 unique contracts with a combined weight of 124360. All the contracts have:

`isOwner(address), revoke(bytes32), hasConfirmed(bytes32,address)`

All but one have:

`addOwner(address), m_numOwners(), changeOwner(address,address), changeRequirement(uint256), removeOwner(address), m_required()` These functions very strongly suggest smart contracts used to record and update ownership. Given the homogeneity and weight of this cluster, an ERC to provide a generic way to record ownership for tokens, tangible and intangible assets, or other contracts is suggested. Optional functions could be defined for different types of

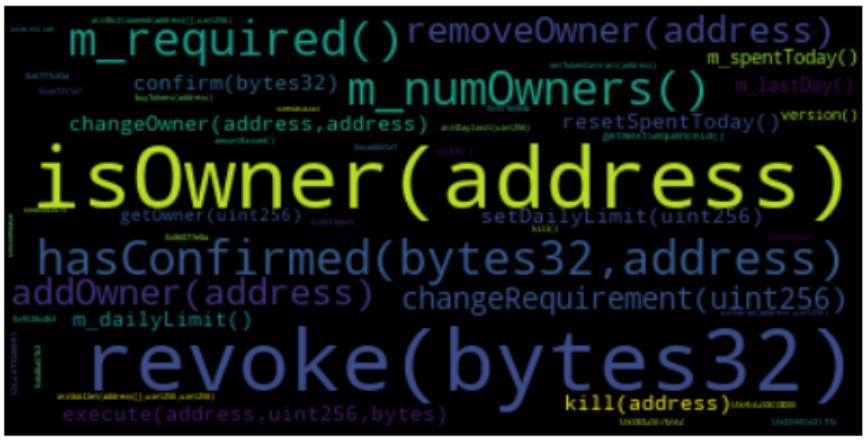


**Figure 7.9** Word cloud of functions in c5 for $k = 9$

asset.

Lastly, c8 contains 2 unique contracts with a combined weight of 5. Although few in number, it is interesting that these contracts were given their own cluster. All the contracts contain the methods: `getMonster(uint256)`, `launchGame()`, `setGameLogicContract(address)`, `spawnMonster(uint256,address)`, `moveToArea(uint16)` On inspection, the functions describe a game. The presence of `withdrawBalance()` in all contracts suggests it may be played for money. While it is not feasible to suggest an ERC for games, which could differ greatly in nature, and require different functions, it is interesting to note that such things have been tried on the Ethereum public chain. If this were a growth area in the future it may be possible to define an extensible ERC for elements common to all games of a given type. For example, all those involving money could be required to define `withdrawBalance()`.

## 7.6 Conclusion

This chapter details the results of clustering Ethereum smart contracts when each contract is treated as the set of the function selectors found in its bytecode. Clusters are labelled according to the results of a reverse look-up, from function selectors to function names. Cluster is carried out using K-means, with different values of k. From $k = 2$ onwards the

major clusters begin to emerge. Accuracy is maintained for increased values of k: $k = 5$ and $k = 9$. New clusters which emerge when increasing the value of k allows for the highlighting of contract purposes for which no ERC currently exists, and make ERC recommendations. The method is validated by showing that ERC20 contracts can be accurately clustered.

As the method is effective it could form the basis of a system used by financial institutions to check the functionality of the contracts they interact with. They would also have the ability to standardise functionality which appears across multiple contracts. Standardisation enables easy identification of contract purpose in the future.

# Chapter Eight

# Conclusions

> "Coming back to where you started is not the same as never leaving."
>
> Terry Pratchett

In this chapter a summary of the work done as part of this thesis, and its contributions is given. Section 8.2 details the future work to be carried out, and why it is important.

## 8.1 Summary

Blockchain can be extremely useful for opening up opportunities for interactions in scenarios where there is a need to enable cooperation. Under certain circumstances it can remove the need for a trusted middle man, or allow parties who do not fully trust one another to do business. The original examples of this are the big public chains like Bitcoin and Ethereum, where currency is exchanged without any central entity processing the transactions. Smart contracts allow for more complex interactions to take place. The entities wishing to transact need only trust the smart contract, and not one another, since smart contract execution cannot be tampered with.

The advantages and new opportunities to trade and interact are further augmented when one considers private chains, where participation can be controlled. Here it is possible to retain the advantages that a blockchain brings, and add the ability to make transactions of a

more sensitive or private nature. In addition, it is possible to use faster consensus algorithms than on a public chain. Blockchains of this kind are extremely useful for KYC data sharing.

Banks can make use of a private chain in order to store an immutable and trusted record of data access permissions. This allows them to share data and also be sure they are in compliance while doing so. Moreover, each entity is accountable for what they request and share, providing a trusted audit trail that can be used by the banks for dispute resolution, or for auditing purposes. Without the blockchain-based solution data sharing would not be possible without heavy reliance on a trusted third party, something that would leave the banks exposed to the risk of increased costs. When coupled with the ability to further incentive sharing by pricing the data, a decentralised, compliant marketplace is created. The architecture of the system is detailed in chapter 5, it provides a decentralised, compliant way to trade data without relying on third parties for storage. Pricing is discussed in chapter 6. The marketplace disrupts the current business models around KYC by creating a new revenue stream for the banks, and removing over reliance on third parties, while also saving time and money for both banks and their customers.

Some of blockchain technology's own technical challenges are considered in this thesis. As blockchains are immutable, the amount of storage required to hold a copy of the chain can only ever increase. With that in mind, chapter 4 details a method to help reduce chain size, this constitutes a contribution in the context of chain performance.

In chapter 6 an exploration is made into the pricing of data within the unique marketplace that the system described in chapter 5 creates. Work with the partner banks enabled the demarcation of different kinds of actors (banks) within the system as buyers and sellers, differing in size and interest. The resultant requirements, in combination with the unique features of the system, are used to contribute a number of pricing mechanisms which would work in the given context for the different types of actors.

Anticipating a future where financial institutions make greater use of smart contracts, it is reasonable to think there may well be future legislation obliging financial entities to

understand and know about the contracts they make use of as part of their business models. Smart contracts are likely to see increased use for interactions between financial institutions, and providing assurances to their customers. In chapter 7 a method for identifying the purpose and functionality of smart contracts using unsupervised learning is presented. As well as proving a method to help identify the purpose of smart contracts, the work done provides a method of finding targets for standardisation of smart contracts. Standardisation could benefit the financial sector as contracts which have a defined structure and set of functions are easier to code, understand and interact with.

## 8.2 Future Work

Future work consists of two different lines of research. Firstly, there are advances to be made in terms testing the system. Chapter 5 details the system designed according the requirements provided by the industry partners. However, there are more features which would benefit from rigorous testing. These include a fully functional file delivery subsystem and scalability testing.

The second line of research to follow is that of the pricing and selling of data. Chapter 6 details various approaches that might be taken in order to price data within the system. Finding the best possible pricing model would be well served by building and testing a number of models. Testing would include both modelling for profit/loss over time, and load testing to ensure data and currency could both changes hands at the required pace. The results of such testing would further enable the partner banks to provide input and feedback on system design and development.

# References


[1] 0x. *ERC-721*. http://erc721.org/. Accessed 1 April 2020.

[2] 4byte. *4byte Directory*. https://www.4byte.directory. [Accessed 11th December 2018]. 2018.

[3] ABBL. "Mutualisation of functions in the Luxembourg Financial Services Section". In: (2019).

[4] Elvira Albert et al. "EthIR: A Framework for High-Level Analysis of Ethereum Bytecode". In: *arXiv preprint arXiv:1805.07208* (2018).

[5] Juan Benet. "IPFS-content addressed, versioned, P2P file system". In: *arXiv preprint arXiv:1407.3561* (2014).

[6] Kumar Bhaskaran et al. "Double-blind consent-driven data sharing on blockchain". In: *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE. 2018, pp. 385–391.

[7] Alex Biryukov, Dmitry Khovratovich, and Sergei Tikhomirov. "Privacy-preserving KYC on Ethereum". In: (2018).

[8] bitcoin. *bitcoin/bitcoin: Bitcoin source tree*. "https://github.com/bitcoin/bitcoin". [Online; accessed 23rd March 2020]. 2020.

[9] Lexi Brent et al. "Vandal: A Scalable Security Analysis Framework for Smart Contracts". In: *arXiv preprint arXiv:1809.03981* (2018).

[10] Vitalik Buterin. *Ethereum Improvement Proposal: Contract code size limit*. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-170.md. [Online; accessed 3rd April 2018]. 2017.

[11] Vitalik Buterin. *State Tree Pruning*. https://blog.ethereum.org/2015/06/26/state-tree-pruning. [Online; accessed 8th March 2018]. 2015.

[12] Brad Chase and Ethan MacBrough. "Analysis of the XRP ledger consensus protocol". In: *arXiv preprint arXiv:1802.07242* (2018).

[13] Ting Chen et al. "GasChecker: Scalable Analysis for Discovering Gas-Inefficient Smart Contracts". In: *IEEE Transactions on Emerging Topics in Computing* ().

[14] Ting Chen et al. "SODA: A Generic Online Detection Framework for Smart Contracts". In: ().

[15] Ting Chen et al. "Towards saving money in using smart contracts". In: *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*. ACM. 2018, pp. 81–84.

[16] Ting Chen et al. "Under-optimized smart contracts devour your money". In: *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*. IEEE. 2017, pp. 442–446.

[17] ConsenSys. *Mithril Classic*. https://github.com/ConsenSys/mythril-classic. [Accessed 8th November 2018]. 2018.

[18] Agencia Espanola Proteccion datos. "INTRODUCTION TO THE HASH FUNCTION AS A PERSONAL DATA PSEUDONYMISATION TECHNIQUE". In: *Official Journal of the European Union 156* (2019).

[19] scikit-learn developers. *sklearn.cluster.KMeans*. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html. [Accessed 16th December 2018]. 2018.

[20] Digiconomist. *Bitcoin Energy Consumption Index*. https://digiconomist.net/bitcoin-energy-consumption. Accessed 3 April 2020.

[21] Christopher Durr. *Parity Hack: How It Happened, And Its Aftermath*. https://medium.com/solidified/parity-hack-how-it-happened-and-its-aftermath-9bffb2105c0. Accessed 1 April 2020.

[22] Ethereum. *Ethereum (ETH) Blockchain Explorer*. https://etherscan.io/. [Accessed 8th November 2018]. 2018.

[23] Ethereum. *Solidity*. https://solidity.readthedocs.io/en/v0.5.1/. [Accessed 16th December 2018]. 2018.

[24] Ethereum. *Solidity 0.6.4 Documentation*. https://solidity.readthedocs.io/en/v0.6.4/. [Release 0.6.4; accessed 24 March 2020]. 2017.

[25] Ethereum. *Vyper Documentation*. https://vyper.readthedocs.io/en/v0.1.0-beta.17/. [Release v0.1.0-beta.17; accessed 24 March 2020]. 2017.

[26] go-ethereum. *clique*. "https://godoc.org/github.com/ethereum/go-ethereum/consensus/clique". [Online; accessed 9th Jan 2020]. 2019.

[27] ethereum. *Remix - Ethereum IDE*. "https://remix.ethereum.org/". [Online; accessed 17th March 2020]. 2020.

[28] ethereum-lists. *List of 4byte identifiers to common smart contract functions*. https://github.com/ethereum-lists/4bytes. [Accessed 13th December 2018]. 2018.

[29] etherum. *ERC | Ethereum Improvement Proposals*. https://eips.ethereum.org/erc. [Accessed 14th November 2018]. 2018.

[30] eth-sri. *eth-sri/securify: Security Scanner for Ethereum Smart Contracts*. https://github.com/eth-sri/securify. [Accessed 9th November 2018]. 2018.

[31] Samuel Falkon. *The Story of the DAO - Its History and Consequences*. https://medium.com/swlh/the-story-of-the-dao-its-history-and-consequences-71e6a8a551ee. Accessed 1 April 2020.

[32] Christof Ferreira Torres et al. "ÆGIS: Smart Shielding of Smart Contracts". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019, pp. 2589–2591.

[33] Michèle Finck. "Blockchain and the General Data Protection Regulation". In: *PE 634.445* (2019).

[34] Ethereum Foundation. *Home | Ethereum Improvement Proposals*. https://eips.ethereum.org/. Accessed 1 April 2020.

[35] Ethereum Foundation. *Home | Ethereum Improvement Proposals*. https://eips.ethereum.org/EIPS/eip-721. Accessed 1 April 2020.

[36] Ethereum Foundation. *Swarm - ethersphere/swarm*. Available at https://github.com/ethersphere/swarm, as of March 2018.

[37] OpenJS Foundation. *Node.js*. "https://nodejs.org/en/". [Online; accessed 17th March 2020]. 2020.

[38] Python Software Foundation. *Python 3.0 Release*. "https://www.python.org/download/releases/3.0/ [Online; accessed 17th March 2020]. 2020.

[39] Samuel A Fricker and Yuliyan V Maksimov. "Pricing of data products in data marketplaces". In: *International Conference of Software Business*. Springer. 2017, pp. 49–66.

[40] Vasilis Gkatzelis, Christina Aperjis, and Bernardo A Huberman. "Pricing private data". In: *Electronic Markets* 25.2 (2015), pp. 109–123.

[41] Golang. *Documentation - The Go Programming Language*. https://golang.org/doc/. Accessed 7 May 2020.

[42] Neville Grech et al. "Madmax: Surviving out-of-gas conditions in ethereum smart contracts". In: *Proceedings of the ACM on Programming Languages* 2.OOPSLA (2018), p. 116.

[43] Nermin Hajdarbegovic. *Bitcoin Miners Ditch Ghash.io Pool Over Fears of 51% Attack*. https://www.coindesk.com/bitcoin-miners-ditch-ghash-io-pool-51-attack. [Accessed 30 March 2020]. 2014.

[44] Hossein Hassani, Xu Huang, and Emmanuel Silva. "Banking with blockchain-ed big data". In: *Journal of Management Analytics* 5.4 (2018), pp. 256–275.

[45] Cameron Huddleston. *50% of Americans Are Cheating — on Their Bank*. https://www.gobankingrates.com/banking/banks/how-many-bank-accounts-americans-have/. [Accessed 15th May 2020]. 2018.

[46] Hyperledger. *Hyperledger Fabric*. "https://www.hyperledger.org/projects/fabric". [Online; accessed 13th Jan 2020]. 2020.

[47] Hyperledger. *Hyperledger Technology Projects*. "https://www.hyperledger.org/projects". [Online; accessed 13th Jan 2020]. 2020.

[48] Hyperledger. *Hyperledger/fabric*. "https://github.com/hyperledger/fabric". [Online; accessed 13th Jan 2020]. 2020.

[49] Bo Jiang, Ye Liu, and WK Chan. "Contractfuzzer: Fuzzing smart contracts for vulnerability detection". In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 2018, pp. 259–269.

[50] Sukrit Kalra et al. "ZEUS: Analyzing Safety of Smart Contracts." In: *NDSS*. 2018, pp. 1–12.

[51] Paraschos Koutris et al. "Query-based data pricing". In: *Journal of the ACM (JACM)* 62.5 (2015), pp. 1–44.

[52] Dapper Labs. *CryptoKitties | About*. https://www.cryptokitties.co/about. Accessed 1 April 2020.

[53] Protocol Labs. *Commands | IPFS Docs*. https://ipfs.io/docs/commands/. [Online; accessed 5th April 2018]. 2017.

[54] Protocol Labs. "Filecoin: A Decentralized Storage Network". In: (2017).

[55] Chao Li et al. "A theory of pricing private data". In: *ACM Transactions on Database Systems (TODS)* 39.4 (2014), pp. 1–28.

[56] Xiao-Bai Li and Srinivasan Raghunathan. "Pricing and disseminating customer data with privacy awareness". In: *Decision support systems* 59 (2014), pp. 63–73.

[57] Xinming Li et al. “Coordinating Data Pricing in Closed-Loop Data Supply Chain with Data Value Uncertainty”. In: *Available at SSRN 3120590* (2018).

[58] Fan Liang et al. “A survey on big data market: Pricing, trading and protection”. In: *IEEE Access* 6 (2018), pp. 15132–15154.

[59] Xueping Liang et al. “Integrating blockchain for data sharing and collaboration in mobile healthcare applications”. In: *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. IEEE. 2017, pp. 1–5.

[60] litecoin-project. *litecoin-project/litecoin: Litecoin source tree*. "https://github.com/litecoin-project/litecoin". [Online; accessed 23rd March 2020]. 2020.

[61] Chao Liu et al. “Reguard: finding reentrancy bugs in smart contracts”. In: *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*. IEEE. 2018, pp. 65–68.

[62] Yvonne Lootsma. “Blockchain as the Newest Regtech Application—the Opportunity to Reduce the Burden of KYC for Financial Institutions”. In: *Banking & Financial Services Policy Report* 36.8 (2017), pp. 16–21.

[63] True Names LTD. *Ethereum Name Service*. https://ens.domains/. Accessed 1 April 2020.

[64] Loi Luu et al. “Making smart contracts smarter”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2016, pp. 254–269.

[65] Damiano Di Francesco Maesa, Paolo Mori, and Laura Ricci. “Blockchain based access control”. In: *IFIP international conference on distributed applications and interoperable systems*. Springer. 2017, pp. 206–220.

[66] Matteo Marescotti et al. “Computing Exact Worst-Case Gas Consumption for Smart Contracts”. In: *International Symposium on Leveraging Applications of Formal Methods*. Springer. 2018, pp. 450–465.

[67] Issac Markus et al. “DAcc: Decentralized Ledger based Access Control for Enterprise Applications”. In: *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE. 2019, pp. 345–351.

[68] Dennis Martens, Alexander van Tuyll van Serooskerken, and Mart Steenhagen. “Exploring the potential of blockchain for KYC”. In: *Journal of Digital Banking* 2.2 (2017), pp. 123–131.

[69] melonproject. “Oyente”. In: (2018). [Accessed 8th November 2018].

[70] Ralph C Merkle. "A digital signature based on a conventional encryption function". In: *Conference on the Theory and Application of Cryptographic Techniques*. Springer. 1987, pp. 369–378.

[71] José Parra Moyano and Omri Ross. "KYC optimization using distributed ledger technology". In: *Business & Information Systems Engineering* 59.6 (2017), pp. 411–423.

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

[73] US NIST. *Descriptions of SHA-256, SHA-384 and SHA-512*. 2001.

[74] Robert Norvill. *ethereum-ipfs-paper-code Private*. https://github.com/pisocrob/ethereum-ipfs-paper-code. [Online; accessed 14th April 2018]. 2018.

[75] Robert Norvill et al. "Automated Labeling of Unknown Contracts in Ethereum". In: (2017), pp. 1–6.

[76] Robert Norvill et al. "Automated labeling of unknown contracts in Ethereum". In: *Computer Communication and Networks (ICCCN), 2017 26th International Conference on*. IEEE. 2017, pp. 1–6.

[77] Robert Norvill et al. "Blockchain for the Simplification and Automation of KYC Result Sharing". In: *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE. 2019, pp. 9–10.

[78] Robert Norvill et al. "IPFS for reduction of chain size in Ethereum". In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE. 2018, pp. 1121–1128.

[79] Robert Norvill et al. "Standardising smart contracts: Automatically inferring ERC standards". In: *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE. 2019, pp. 192–195.

[80] José Parra-Moyano, Tryggvi Thoroddsen, and Omri Ross. "Optimized and dynamic kyc system based on blockchain technology". In: *Available at SSRN 3248913* (2018).

[81] Colin Percival and Simon Josefsson. "The scrypt password-based key derivation function". In: *IETF Draft URL: http://tools. ietf. org/html/josefsson-scrypt-kdf-00. txt (accessed: 30.11. 2012)* (2016).

[82] Beltran Borja Fiz Pontiveros, Robert Norvill, and Radu State. "Recycling Smart Contracts: Compression of the Ethereum Blockchain". In: *New Technologies, Mobility and Security (NTMS), 2018 9th IFIP International Conference on*. IEEE. 2018, pp. 1–5.

[83] Joseph Poon and Vitalik Buterin. "Plasma: Scalable Autonomous Smart Contracts". In: *White paper* (2017).

[84] Joseph Poon and Thaddeus Dryja. "The bitcoin lightning network: Scalable off-chain instant payments". In: *draft version 0.5* 9 (2016), p. 14.

[85] Open Bank Project. *API Explorer*. "https://apiexplorersandbox.openbankproject.com/?version=2.0 [Online; accessed 17th March 2020]. 2020.

[86] Peter Reville. "ATM Banking: It's Not Just About Cash Withdrawal Anymore". In: *Mercator Advisory Group - North American PaymentsInsights* (2019).

[87] David Schwartz, Noah Youngs, Arthur Britto, et al. "The ripple protocol consensus algorithm". In: *Ripple Labs Inc White Paper* 5.8 (2014).

[88] Wazen Shbair, Mathis Steichen, Jérôme François, et al. "Blockchain orchestration and experimentation framework: A case study of KYC". In: 2018.

[89] Michael Siegenthaler and Ken Birman. "Sharing private information across distributed databases". In: *2009 eighth ieee international symposium on network computing and applications*. IEEE. 2009, pp. 82–89.

[90] Mathis Steichen et al. "Blockchain-Based, Decentralized Access Control for IPFS". In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE. 2018, pp. 1499–1506.

[91] Clare Sullivan and Eric Burger. "E-residency and blockchain". In: *Computer Law & Security Review* 33.4 (2017), pp. 470–481.

[92] Michael J Swain and Dana H Ballard. "Color indexing". In: *International journal of computer vision* 7.1 (1991), pp. 11–32.

[93] Péter Szilágyi. *eth/63 fast synchronization algorithm*. https://github.com/ethereum/go-ethereum/pull/1889. [Online; accessed 8th March 2018]. 2015.

[94] CryptoKitties Team. "CryptoKitties: Collectible and Breedable Cats Empowered by Blockchain Technology". In: *Self Published* (2020).

[95] Coil Technologies. *Coil - About*. https://coil.com/about. Accessed 1 April 2020.

[96] Stefan Thomas and Evan Schwartz. *White Paper*. https://github.com/codius/codius-wiki/wiki/White-Paper. Accessed 1 April 2020.

[97] trailofbits. *rattle: static evm binary static analysis*. https://github.com/trailofbits/rattle. [Accessed 8th November 2018]. 2018.

[98] Nguyen Binh Truong et al. "GDPR-Compliant Personal Data Management: A Blockchain-based Solution". In: *arXiv preprint arXiv:1904.03038* (2019).

[99] Petar Tsankov et al. "Securify: Practical Security Analysis of Smart Contracts". In: *arXiv preprint arXiv:1806.01143* (2018).

[100] European Union. "DIRECTIVE (EU) 2018/843 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL". In: *Official Journal of the European Union 156* (2018), pp. 43–74.

[101] European Union. "REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL". In: *Official Journal of the European Union 119* (2016), pp. 1–88.

[102] usyd-blockchain. *Vandal*. https://github.com/usyd-blockchain/vandal. [Accessed 8th November 2018]. 2018.

[103] Fabian Vogelsteller and Vitalik Buterin. *ERC-20 Token Standard*. https://eips.ethereum.org/EIPS/eip-20. [Accessed 8th November 2018]. 2015.

[104] Weina Wang, Lei Ying, and Junshan Zhang. "Buying data from privacy-aware individuals: the effect of negative payments". In: *International Conference on Web and Internet Economics*. Springer. 2016, pp. 87–101.

[105] Bitcoin Wiki. *Confirmation*. https://en.bitcoin.it/wiki/Confirmation. [Accessed 30 March 2020]. 2018.

[106] Gavin Wood. "Ethereum: A secure decentralised generalised transaction ledger". In: *Ethereum Project Yellow Paper* 151 (2014).

[107] Qi Xia et al. "BBDS: Blockchain-based data sharing for electronic medical records in cloud environments". In: *Information* 8.2 (2017), p. 44.

[108] Joseph Young. *CryptoKitties Sales Hit $12 Million, Could be Ethereum's Killer App After All*. https://cointelegraph.com/news/cryptokitties-sales-hit-12-million-could-be-ethereums-killer-app-after-all. Accessed 1 April 2020.

[109] Guy Zyskind, Oz Nathan, et al. "Decentralizing privacy: Using blockchain to protect personal data". In: *2015 IEEE Security and Privacy Workshops*. IEEE. 2015, pp. 180–184.