A Security and Privacy Focused KYC Data Sharing Platform

Robert Norvill
robert.norvill@uni.lu
Sedan Group, SnT, University of Luxembourg, Luxembourg
Luxembourg

Cyril Cassanges
cyril.cassagnes@uni.lu
Sedan Group, SnT, University of Luxembourg, Luxembourg
Luxembourg

Wazen Shbair
wazen.shbair@uni.lu
Sedan Group, SnT, University of Luxembourg, Luxembourg
Luxembourg

Jean Hilger
jean.hilger@ext.uni.lu
Banque et Caisse d'Epargne de l'Etat Luxembourg, Luxembourg

Andrea Cullen
a.j.cullen@bradford.ac.uk
EECS - Engineering & Informatics, University of Bradford, Bradford
United Kingdom

Radu State
radu.state@uni.lu
Sedan Group, SnT, University of Luxembourg, Luxembourg
Luxembourg

ABSTRACT

Banks in Europe must comply with new EU regulation and legislation. Recent legislation has focused on personal data, Know Your Customer (KYC), and anti-money laundering. As a result, the cost of KYC compliance is higher than ever, requiring time consuming work by both the banks and their customers in the form of document collection and verification. In this paper we detail a system designed to ease the burden of compliance for banks within the EU and save their customers time through the secure and permissioned sharing of digital KYC data. In order to share data, banks need a secure system capable of protecting the privacy of both them and their clients. We detail a system which uses blockchain technology and various privacy and security enhancing techniques to provide banks with a fast and secure way to share documents required for know your customer compliance. The system was built to be aligned with the GDPR, meaning each participating bank must have explicit permission for a customer to access one or more of their documents. These permissions are stored on a private blockchain shared by the banks. Moreover, we detail methods to anonymise on-chain data where necessary. The use of a private blockchain to achieve consensus on the veracity of customer-granted permissions to data enables participating banks to trust one another as each permission and request is observed, agreed upon, and stored on-chain. To the best of our knowledge we propose the first data sharing system under which there is no outsourcing of data storage. This allows the banks to retain full control of storage security and encryption.

KEYWORDS

data sharing, blockchain, KYC, privacy, security, anonymity

BSCI '20, October 6, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-7610-5/20/10…$15.00
https://doi.org/10.1145/3384943.3409431

ACM Reference Format:
Robert Norvill, Cyril Cassanges, Wazen Shbair, Jean Hilger, Andrea Cullen, and Radu State. 2020. A Security and Privacy Focused KYC Data Sharing Platform. In *Proceedings of the 2nd ACM International Symposium on Blockchain and Secure Critical Infrastructure (BSCI '20), October 6, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3384943.3409431

1 INTRODUCTION

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 activities and verification work carried out by banks. From a client perspective, the lack of data sharing means that using the services of another bank requires the numerous 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 of their customers caused by the need to repeatedly present documentation to the banks, and ensure banks fully comply with KYC/GDPR legislation while sharing data. The 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 paper the 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 we detail is designed for banks in the EU. They are legally mandated to comply with KYC and anti-money laundering (AML) legislation, and the recently introduced General Data Protection Regulation (GDPR). As such, banks now have more compliance requirements than ever before. In this work we focus 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. We do this by allowing banks to share customer data between one another. Through doing so we enable the banks to reduce their costs and save time for their

clients by introducing a model of interaction between banks which did not previously exist. Secondly we focus on issues surrounding the privacy and security of KYC personal data, especially when transferring it between banks. GDPR requires that the 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 right the client also has the right to request the deletion of their data, other legislation not withstanding. Our system ensures GDPR compliance be storing explicit permissions to access customer data on the distributed private blockchain. Documents are never stored on-chain, in addition we make use of cryptographic techniques such as salted hashing to ensure a client's request for deletion can always be complied with.

Banks have various different kinds of customers, all of which can benefit from our system. We categories customers 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, our 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 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, based on their consent. 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 there is 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 store in their internal encrypted storage and that they have control over the data flow between their internal systems and the KYC sharing system.

Our 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 2. These include blockchain-backed data access permissions to control the flow of data between custodians, ensuring client privacy and anonymity is maintained within the system. The use of pseudonymous (random identifiers) on-chain to protect privacy. Salted hashing is used where appropriate to allow for anonymisation of identifiers. Lastly, we use container technology to segregate the bank's databases from the data sharing system, all of a bank's data is retained by them individually.

The problem we aim to solve with the system detailed in this paper is directly defined by industry requirements. We work very closely with our industry partners in the Luxmebourgish Bankers Association (ABBL). Within the ABBL we work with three partner banks that provide our 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 and KYC compliance, and standardisation hold the top three spots for processes banks believe it would be beneficial to mutualise [1]. In this paper we propose a P2P, blockchain-backed system as a solution, with the banks as peers. 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.

Our 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 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 main contributions of this work are as follows: 1) We detail a blockchain based data sharing system, designed 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, we present results showing how the proposed solution performs under stress loading.

The rest of this paper is structured as follows; in section 2 we detail the requirements of our use case as guided by our real world scenario and industry partners. In section 3 we discuss the major blockchain systems and works that have applied them in order to create data sharing systems. Section 4 we detail our system and its functionality. Section 5 details the technical implementation of the system. In section 6 we detail the tests carried out and interpret the results of these tests. Finally, sections 7 and 8 provide the conclusion and future work, respectively.

2 REQUIREMENTS

In this section we detail each of our requirements by identifying the needs of our pilot banks, and then detailing how our system aims to meet these needs. Each requirement is a result of the major challenges surrounding KYC.

Reduction of costs: Our 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 valid 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.

We aim to meet this need through data sharing. 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 our system is reduction of costs though 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, it is necessary that data is accurate, and providers are accountable. The banks need be able to trust in the integrity of the data they receive, and customers require 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 banks to the risk of becoming overly dependent on one entity, which could incur them great expense, or expose them to legal costs 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, we utilise blockchain to act as a decentralised, immutable record of data access permissions, observed and agree 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 our industry partners, banks must retain all data bout their clients, and not rely on each other, or third-party solution for storage of data.

Our 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 4.2. 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, we give the customer 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 information is stored on the chain in order to ensure alignment with GDPR. We make use of salted hashes in order to decouple the hash from the personal data used to generate it. Through doing so we are able 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 on-chain hashed data, 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 to integrate a new system do to the need 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. We make use of containerisation in order to make it easy for banks to run the required software. We require only that banks install the docker client software, which has already seen wide spread adoption in the banking sector. This helps us ensure we meet our requirement that the solution can be deployed as frictionlessly as possible, with minimum disruption to the bank's existing systems.

3 RELATED WORK

There are now a number of well known and widely utilised blockchain systems and frameworks in existence. As we carry out access control checks on-chain we deal with blockchain software that is capable of executing code on-chain through the use of smart contracts. The grandfather of cryptocurrencies, Bitcoin, is capable of limited on-chain code execution. However, it's functionality is restricted by design to avoid attacks that could slow down or otherwise exploit the system. Details of the original Bitcoin system can be found in the white paper [15]. A description of the script that can be used in transactions can be found at [20].

Ethereum took a different approach to mitigating the risk of arbitrary on-chain code execution and provides a set of instructions (opcodes) that can be used to create smart contracts, usually a higher level language is compiled to opcodes. Users pay in 'gas' per-opcode for execution to deter unnecessary or lengthily executions, a gas limit is imposed, which execution cannot exceed. As problematic execution is restricted in this way, an honest user has the scope to write much more complex and useful contracts for Ethereum. We leverage contracts to provide data access control between banks. Details of contract execution and the opcodes can be found in the yellow paper [21].

What makes Ethereum still more attractive for our system is the fact that it is built with the ability to be run as a private permissioned blockchain, and provides ready-to-use Proof of Authority (PoA) consensus, which is much faster and less computationally expensive. It is well suited to a private-chain environment such as that required by our system. The documentation for the go-ethereum (geth) implementation of the clique PoA consensus algorithm can be found here [3].

Under Hyperledger as an umbrella project there are currently 10 distributed ledger systems [9]. One of these systems is Hyperledger Fabric, which is an open source project that provides a framework for building distributed ledgers [10]. 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 [8].

Various academic works exist that utilise blockchain for access control. In [11] 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 price their services. However, the use of cloud storage for the collected data runs contrary to the requirement of our partner banks that they retain all their customer's 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 [22], they also make use of a centralised data storage solution.

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. They provide a formal definition of the protocols they created for access control [23].

Gai et al. propose a private blockchain based system for trading energy. In addition to enabling energy trading, the authors detail methods for shielding the user's privacy by obfuscating information which could be data mined to build up a picture of the persons trading within the system.

In [12] 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 our scenario, of being slower and more expensive, as well as transactions being publicly visible.

In [19] Troung et al. detail a GDPR-compliant, blockchain based system for data management. Like our solution, 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", under our system each bank retains the data for each of its clients.

Similarly, in [7] Gai et al. details an Ethereum based system whereby blockchain technology is used for data governance in an edge computing environment. The system the authors propose a focuses on privacy preservation and aims to minimise the impact on power consumption for IoT devices.

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 [13]. Such a storage solution results in data being stored on behalf of other entities, something thatis necessary to avoid in order to meet our requirements.

In [14] Moyano et al. detail a distributed ledger based system to help reduce the costs of KYC processes for financial institutions. They highlight the same issue as this paper; KYC processes are replicated per bank. They similarly propose a distributed ledger with the banks as nodes, where clients giving permission to share their data between banks. Their assumptions differ in that they assume all banks will be in the same nation, and the system will be maintained by the national regulator. On the other hand we assume a bank can be anywhere in the EU, with the system being maintained by the member banks and 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. The major differences between the paper in question and ours are that we increase the scope from national to international, and lessened burden on the regulator by removing the reliance on third-party storage.

Lastly in our previous paper 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 [18].

4 DESIGN

In this section we detail the design of our system and how it meets the requirements listed in section 2. We look at, in order, This includes the implementation and use of: containerisation technology, a private Ethereum blockchain, a smart contract, and API's.

4.1 Containerisation

One of our partner banks' requirements is that integration should be as frictionless as possible. We require only that banks install the single piece of software needed to run the containers, and implement functionality for our API to interface with the bank's internal systems. This is large a reduction in work on the bank's part, compared to the full cycle of 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. 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 houses a logically separate part of system. This allows us to logically separate out different functionalities. Through doing so we allow 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 us to implement data segregation, we can 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. This is demonstrated in figure 1 which shows specific directors being mapped from the container's file system to the host Operating System. Using this model for persistent storage we are able to ensure that containers only access the locations relevant to their operations. Moreover, the container responsible for data transfer must request data from the bank's internal 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 3. 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 3 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.

4.2 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

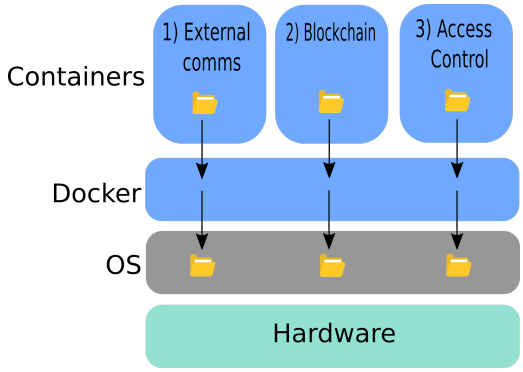

Figure 1: Vertical slice view of containerisation and mapping of directories

each of its customers. This was a hard requirement for our industry partners, as such the system we detail in this paper 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.

The data storage model of our system can be seen by way of and example in figure 2. 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 retrieve's 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 blockchain 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 we cannot store any personal documents on-chain. The on-chain identifiers take the form of randomly generated ID's for clients, and salted hashes for documents. Both the ID and the hash is designed to be separable from the client, that is to say, both can be made anonymous, in alignment with GDPR. In both cases each 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 it 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

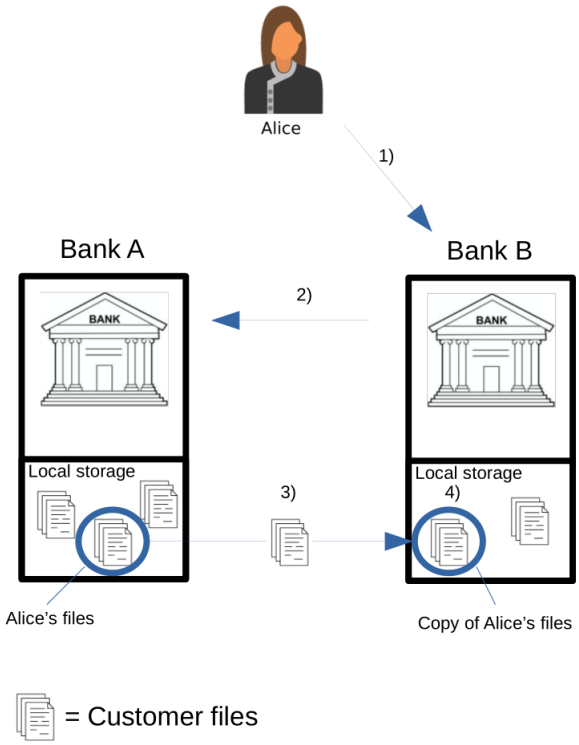

Figure 2: High level overview of document acquisition process for a customer beginning a relationship with their second bank

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 associated the relevant salt with it. The hash cannot be generated without both the document and the hash. 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 it cannot recreate it, ensuring GDPR compliance.

4.3 Data Flow

To give participating banks the highest possible levels of assurance about when and with which entities personal data can be shared our 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 our 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 than can withhold their response to it.

4.4 External Communications Container

Container 1 in figure 3 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 geographical 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. This container can be augmented to include further automated checks and services that the banks would like to use. Here we enhance trust in the data as any check handled by the container is carried out in the exact same way regardless of which bank does it.

4.5 Blockchain Platform Container

Container 2 in figure 3 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 data is only shared when the permission to do so is given. In this way we control the flow of data between banks in order to meet our trust and privacy requirements. Container 3 in figure 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 our working prototype runs with both Ethereum's geth implementation and Hyperledger Fabric depending on which container is connected.

4.6 Data Access Control System Container

Lastly, with regard to figure 3, 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 (container 4), to request the necessary customer documents. Here we 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, 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.

4.7 The Smart Contracts

Our system design includes two Smart Contracts. First, the access control smart contract records client permissions for banks to access their data. The second contract facilitates 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 which information. 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 permissions process for on-boarding is as follows:

(1) Customer asks their current bank to generate a single use authentication pass-code.
(2) The current bank generates the pass-code as well as a hash of it.
(3) The current bank sends the hash of the pass-code and the customer's ID to the on-boarding contract.
(4) The on-boarding contract stores the two pieces of information and associates them using mappings.
(5) The customer presents the pass-code to the new bank.
(6) The new bank sends the pass-code to the on-boarding contract.
(7) The on-boarding contract generates the hash and checks if a valid pairing using the hash exists.

(8) If a match exists the bank can use the associated customer ID to make requests for data.

If the pass-code is ever lost or leaked, the current bank can, upon receiving notification from the client, send a follow-up transaction to the contract declaring that the pass-code (and its hash) void, and issue a new pass-code to the client repeat to process of association. The on-boarding permissions process ensures that clients retain one ID across all banks.

4.8 Data Pricing

In order to make our system more attractive to banks, and further alleviate the costs associated with KYC we cover in brief the remuneration for banks which carry out KYC processes, under our system, the bank that does the original KYC incurs the costs of collection and/or verification is paid by other banks which request the data. The full pricing system is detailed in [16].

To further mitigate the costs of KYC, specifically those incurred from document collection and verification, we augment out system with a pricing model whereby the bank that collects and verifies a document is able to trade it with permissioned banks in order to recover their costs.

Data is non-rivalrous, meaning it being sold to one entity does not reduce the quantity and it can be sold again. In the case of KYC data the average natural person customer often has a relationship with multiple banks and the documents they provide to each bank are likely to be similar (ID card, proof of address, etc.). We can divide the cost of document acquisition by the number of expected sales: $p = c/n$. Where p is the price of sale for a document, c is the cost of acquisition and verification, and n is the number of expected purchases based on client type. It is also possible to include a percentage profit margin in the formula: $p = c * (1 + x)/n$ where x is the desired profit margin expressed as a value between 0 and 1. The same formula applies to natural persons and corporate entities.

Such a pricing model has the advantage of being simple to implement with smart contracts, enabling the lowering of the costs of data acquisition. Thanks to the high likelihood that data is sold multiple times, cost is also reduced for purchasing banks. A smart contract can use be used to facilitate the trade, utilising a token that is tied 1:1 to the banks' currency of choice, in our case, the Euro. The contract takes the determined amount in tokens from the buyer and hold them until the buyer confirms receipt of the data, at which point the tokens are be release to the sender.

5 IMPLEMENTATION

In this section we provide 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 [5]. We focus on the Ethereum based implementation of our system. Scripts for first-time setup, used for configuring geth and launching the smart contracts, are written in Python3 [6].

5.1 Communications

As mentioned in section 4 all parts of the system are containerised. We utilise Docker 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 we use is an extension of the PSD2 compliant open bank project API [17], it is designed for the acquisition of customer information. We extend it in order to provide some of specific inter-container communication. Each container in the system contains a subset of our full API, depending on the tasks it is required to carry out. The API is written in Node.js.

5.2 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 [4]. 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 structs are used in order to store file hashes associated with a 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.

5.3 Blockchain Platform Container

Our system makes use of a private, permissioned Ethereum chain with each bank running the go-lang Ethereum client software (geth) [2]. Each bank has one verifier address, creating a 'one bank one vote' system. We make use of Ethereum's PoA implementation, clique, as our consensus mechanism [3]. 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. We make use of the puppeth, a CLI tool that comes bundled with geth 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 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 our system salts are randomly generated strings. Each file has its own separate, unique

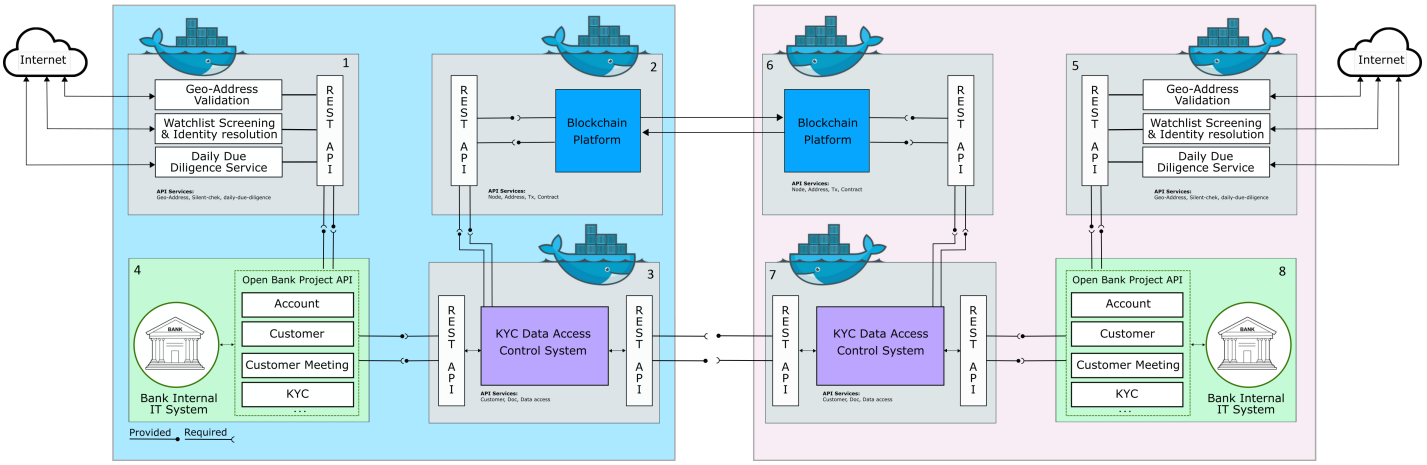

Figure 3: System Architecture for Two Banks Running the System

salt. We use the SHA3 algorithm to generate strong cryptographic hashes. Each salt is stored by the bank in their interal storage system and associated with the relevant file. The hash is stored on-chain and associated with the 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.

5.5 The DAC Container and file Transfer

Next we take a closer look at how file transfer is handled by our 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 fullfilling 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.

5.6 Deployment

Here, we look at the technical requirements for banks implementing the system, of which there are two. We require that that the banks install docker, such that they are able to run the containers. Banks must implement part of our 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 1: Throughput and latency1.5

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	0.60	0.32	13.33	22.5

Table 2: Load tests and their operations

Test	Functionality	Transactions	Local chain reads	HTTP requests
Add File	Add 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

6 RESULTS

In this section we detail the results of load testing a 3 instance setup of the blockchain based functionality of our system, namely the DAC and BP containers. We run tests 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 communications between the local DAC and BP containers. In turn the BP containers of each instance communicate with one another to perform on-chain operations which propagate throughout the network. We built a network of 3 instances of our system, running in disparate global locations: California, Paris, Tokyo. All instances are connected to each other and all run an docker 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.

We run 5 different tests, they replicate the kind of operations that would be carried out by banks utilising the system. They are detailed in table 2 in order of the increasing complexity. Each test is run for 100 seconds with requests sent in parallel; 10 at a time with 2 second intervals after each request. Each request waits to receive the appropriate response from the DAC container, we record how long this takes 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 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 our system is capable of on-boarding and performing complex on-chain operations for 10 customers simultaneously. Average on-boarding time is 1.5 customers per second. Moreover, the results 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 3.16 seconds. This test carries out these operations much faster than a customer is likely to request them in a real-world scenario, showing that the system is capable of handling even a large influx of customers with complex operational requirements.

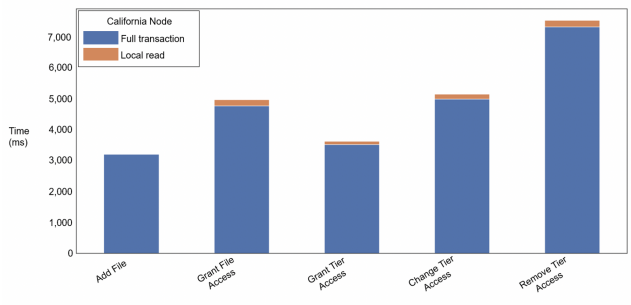

Figure 4: Average chain interaction times for tests on California server

As can be expected, local read-only operations on-chain-data are much faster than full blockchain transactions. Geth 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. We leverage the ability to make local calls by making as many transactions as possible stateless. Local calls are used in any 'check' function (check file access, for example). On our Paris instance the average time between sending a HTTP request to check file access and receiving a response during the Remove File Access test was 39ms, whereas the average for adding a file (a full blockchain transaction) was 3698ms. The difference between the time taken to achieve consensus between internationally located instances under a high transaction volume, and to access a local copy of the chain is 3659ms. 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 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 4 and figure 5 for California and Paris, respectively.

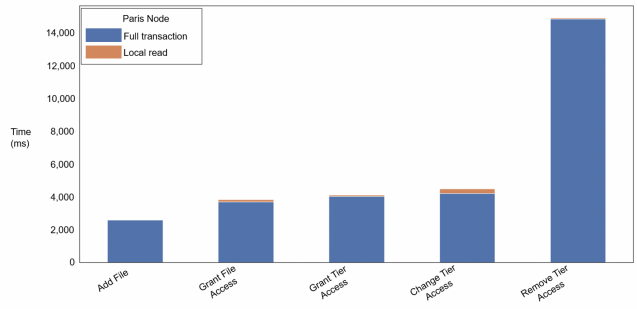

Figure 5: Average chain interaction times for tests on Paris server

7 CONCLUSION

We detail 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 our industry partners. The system meets the needs of our industry partners, including requirements including not relying on centralised third-parties for storage or data acquisition. We load test the system 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.

8 FUTURE WORK

Future work includes running larger scale tests with a larger network, with requests coming from multiple instances at the same time. Moreover, we aim to deploy the system on partner banks' infrastructure to ascertain system functionality in full operational use.

9 ACKNOWLEDGEMENTS

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 830927.

REFERENCES

[1] ABBL: Mutualisation of functions in the luxembourg financial services section (2019)
[2] go-ethereum Authors, T.: Geth documentation. "https://geth.ethereum.org/docs/" (2020), [Online; accessed 14th July 2020]
[3] go ethereum: clique. "https://godoc.org/github.com/ethereum/go-ethereum/consensus/clique" (2019), [Online; accessed 9th Jan 2020]
[4] ethereum: Remix - ethereum ide. "https://remix.ethereum.org/" (2020), [Online; accessed 17th March 2020]
[5] Foundation, O.: Node.js. "https://nodejs.org/en/" (2020), [Online; accessed 17th March 2020]
[6] Foundation, P.S.: Python 3.0 release. "https://www.python.org/download/releases/3.0/" (2020), [Online; accessed 17th March 2020]
[7] Gai, K., Wu, Y., Zhu, L., Zhang, Z., Qiu, M.: Differential privacy-based blockchain for industrial internet-of-things. IEEE Transactions on Industrial Informatics **16**(6), 4156–4165 (2019)
[8] Hyperledger: Hyperledger fabric. "https://www.hyperledger.org/projects/fabric" (2020), [Online; accessed 13th Jan 2020]
[9] Hyperledger: Hyperledger technology projects. "https://www.hyperledger.org/projects" (2020), [Online; accessed 13th Jan 2020]
[10] Hyperledger: Hyperledger/fabric. "https://github.com/hyperledger/fabric" (2020), [Online; accessed 13th Jan 2020]
[11] Liang, X., Zhao, J., Shetty, S., Liu, J., Li, D.: 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). pp. 1–5. IEEE (2017)
[12] Maesa, D.D.F., Mori, P., Ricci, L.: Blockchain based access control. In: IFIP international conference on distributed applications and interoperable systems. pp. 206–220. Springer (2017)
[13] Markus, I., Xu, L., Subhod, I., Nayab, N.: Dacc: Decentralized ledger based access control for enterprise applications. In: 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). pp. 345–351. IEEE (2019)
[14] Moyano, J.P., Ross, O.: Kyc optimization using distributed ledger technology. Business & Information Systems Engineering **59**(6), 411–423 (2017)
[15] Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
[16] Norvill, R., Hilger, J., Awan, I., Cullen, A., State, R.: Decentralised compliant data trading for banks. In: 2020 ACM International Electronics Communication Conference. ACM (2020)
[17] Project, O.B.: Api explorer. "https://apiexplorersandbox.openbankproject.com/?version=2.0.0" (2020), [Online; accessed 17th March 2020]
[18] Steichen, M., Fiz, B., Norvill, R., Shbair, W., State, R.: 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). pp. 1499–1506. IEEE (2018)
[19] Truong, N.B., Sun, K., Lee, G.M., Guo, Y.: Gdpr-compliant personal data management: A blockchain-based solution. arXiv preprint arXiv:1904.03038 (2019)
[20] Wiki, B.: Script. "https://en.bitcoin.it/wiki/Script" (2019), [Online; accessed 9th Jan 2020]
[21] Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper **151** (2014)
[22] Xia, Q., Sifah, E., Smahi, A., Amofa, S., Zhang, X.: Bbds: Blockchain-based data sharing for electronic medical records in cloud environments. Information **8**(2), 44 (2017)
[23] Zyskind, G., Nathan, O., et al.: Decentralizing privacy: Using blockchain to protect personal data. In: 2015 IEEE Security and Privacy Workshops. pp. 180–184. IEEE (2015)