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

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

Jean Hilger
jean.hilger@ext.uni.lu
Banque et Caisse d'Épargne de l'État
Luxembourg

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

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

ABSTRACT

Banks in Europe must comply with new EU regulation and legislation. Recent legislation has focused on personal data, know your customer, and anti-money laundering. The cost of know your customer compliance 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 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. In this paper 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.

KEYWORDS

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

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 *ASIACCS '20: ACM ASIA Conference on Computer and Communications Security, June 01–05, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

ASIACCS '20, June 01–05, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

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 Know Your Customer (KYC) checks. However, at present banks do not normally share their KYC data, or the results of their KYC related document verifications and checks. This is due to each bank being legally responsible for any data it holds, and the fact that banks are market place rivals. This leads to a huge duplication of document acquisition activities and KYC work being carried out by banks. From a client perspective, 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, where 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 the repeated giving of documents for their customers, and comply with KYC/GDPR legislation. 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 they must be sure 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.

Banks in the EU are legally mandated to comply with know your customer (KYC) and anti-money laundering (AML) as well as the recently introduced General Data Protection Regulation (GDPR). As such, banks now have more compliance requirements than ever before. We focus on two major issues surrounding KYC. Firstly, the time and expense incurred by carrying out KYC processes. Secondly, issues surrounding the privacy and security of KYC personal data, especially when transferring it between banks (data custodians). GDPR requires that the client (the data owner) controls their own data, including for what purposes it can be used, and by which entities. It also encompasses the right for the client 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.

We consider customers to be 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 security and privacy assurances by legal persons and corporates to control the flow of any data they deem sensitive, and to ensure its security. In doing so 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 automatically in compliance with KYC legislation in most cases as they can automatically receive updates for data which they have permission to access. Banks are assured that the risk of a data breach is minimised due to the segregation of their internal encrypted storage and the KYC sharing system.

Our system was built to provide 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.

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 a snap 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 agreed upon, 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.

2 REQUIREMENTS

With KYC, GDPR, and AML laws, banks have greater compliance requirements in Europe 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 terrorists 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 corporates the documentation can 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. We aim to reduce some of the associated costs through data sharing. A such we address the security and privacy concerns surrounding the sharing of private data. Below we detail the major challenges that our system aims to provide a solution to.

Trust: Each bank is legally responsible for the validity of the KYC documents they hold. For being able to exchange data with confidence in its accurate accountability is key. 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. We utilise blockchain to act as an 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.

Data privacy & pseudonymisation: As distributed ledgers 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. 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 anonymous 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. The immutable on-chain record for access control ensures that the customer is in control of their data.

Data segregation & flow: 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 containerised system's interaction is entirely at the discretion of the host, with the container not having access to any directories other than those explicitly granted to it. The mapping of directories in containers to specified host directories is depicted in 1. In our system any data passed to the system must first be requested via an internal API call to the hosting bank. The bank is then free to make their own decisions on responding to the call. This method of segregation meets the requirements of the banks that they are in control of their data at all times. In short, the system has no access to any stored private data except via an API call which the bank can response to. 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.

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 [11]. A description of the script that can be used in transactions can be found at [14].

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 [15].

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 [5]. One of these systems is Hyperledger Fabric, which is an open source project that provides a framework for building distributed ledgers [6]. 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 [4].

Containerisation is often utilised to ensure that software can run across heterogeneous operating systems and hardware. It is generally OS-level virtualisation, which is much less resource intensive than full virtual machines. It achieves this by requiring only that the target system has the client-side container software installed, as oppose to all languages/modules/packages/pieces of software that the containerised system requires. All these things are packed inside a container which can then be run on any system which has the client-side software installed. In terms of security, due to the level of virtualisation, the container can only see and interact with directories on the host system which it is explicitly given access to, meaning the system cannot see or touch anything outside of the directories it uses to store the blockchain and record necessary data (e.g. logs). Moreover, the system only has access to the personal data stored by the bank through API requests, meaning the bank is in full control of any data leaving its internal systems. The ability to run across different systems, restricted access to the host system and the request-only nature of data access are very positive things as banks often have very different internal IT solutions. We make use of Docker, a prominent and well supported containerisation system to allow us to meet the requirements of our partner banks [2].

Various academic works exist that utilise blockchain for access control. In [7] 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 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 [16].

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 [17].

In [8] 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 [13] 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 out system each bank retains the data for each of its clients.

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 [9].

In [10] 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. The 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.

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 [12].

4 THE SYSTEM

In this section, we first detail the design and architecture of our system, and how this helps to meet its requirements. In section 4.2

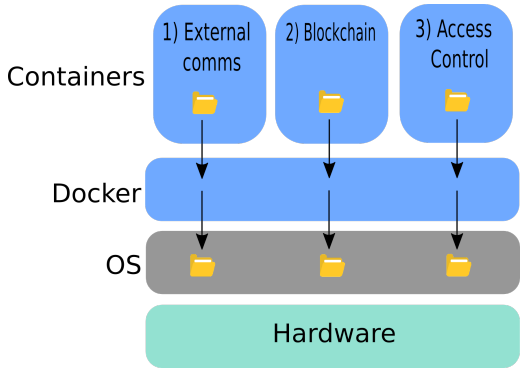

Figure 1: Vertical slice view of containerisation and mapping of directories

we give details about the implementation of the prototype built and deployed for testing.

4.1 Design

The system's functionality is spread across three containers with each container housing 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. Containers are under the control of the host, meaning 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. 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 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 6-7 for the second bank. Figure 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 the containers the explanations apply to every participating financial institution.

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 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.

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 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 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. Data privacy is further assured by the fact that only pseudonymous identifiers for documents are stored on-chain as salted hashes. Under GDPR pseudonymous data is also considered personal data. We use deletion of off-chain data and salts to anonymise the on-chain identifiers. 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 (geth) and Hyperledger Fabric depending on which container is connected.

Lastly, container 3 interacts with container 2, it also operates a white list. Upon receiving a data request, container 3 will wait on the response from 2 before making a call to the bank's internal systems (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 controlled by a pull, and not a push, model 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 simple 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, as containers only interact using their specific API's. Secure data management is ensured here as at each stage of the process for sending data 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

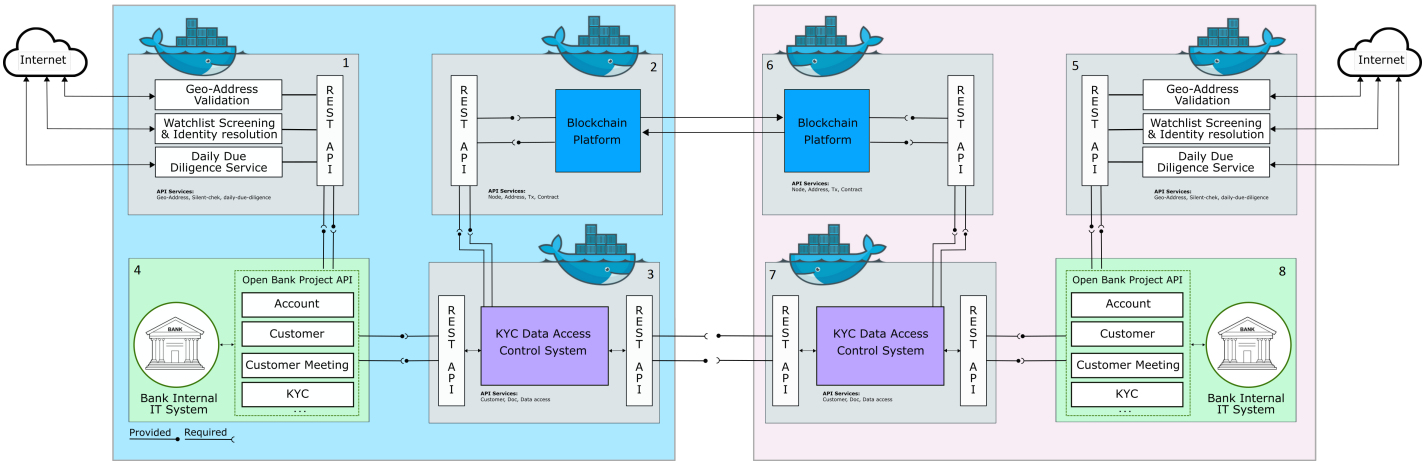

Figure 2: System Architecture

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.

4.2 Technical Details

In this section we provide details of specific technologies and methods used to help ensure privacy and data security.

Consensus: Consensus among peers is necessary for the ledger to be trusted. Our system makes us of a permissioned private Ethereum chain with each bank running the go-lang Ethereum client software (geth). Each bank has one verifier address, giving a 'one bank one vote' system. For the tests described in section 5 Ethereum's PoA implementation, clique is used.

Privacy: In order for our system to be aligned with GDPR we cannot store any personal documents on-chain, as the chain is immutable and we must retain the ability to delete or anonymise data. As such, all documents are stored in the bank's internal storage systems. This allows for ease of deletion, and also meets the banks' requirement that they own and store all of the data about their clients. Under GDPR on-chain identifiers likely count as pseudonymous data, which must be removed or anonymised in the event of a request for deletion. On-chain identifiers take the form of randomly generated ID's for clients, and salted hashes for documents. In both cases anonymisation is achieved by the deletion of off-chain records. For random ID's, each bank holds mappings between the random ID and the client identity for all their clients. In the event that deletion is legitimately requested the bank needs only to erase the mapping for it to become impossible to ascertain to whom the ID belonged. In the case of salted hashes as document identifiers, each banks holds records of which salt is used for which file, when a file is shared with a bank it also receives the salt. In the event of a request for deletion the bank must delete the document and the salt. It is not possible to re-create the hash, and thus link it to the client, even if one is in possession of the document, as the salt will still not be known.

5 RESULTS

In this section we detail the results of load testing the blockchain based functionality of our system. The tests involve the Data Access Control (DAC) container (3) and the Blockchain Platform (BP) container (2). We utilise the k6 performance testing tool to generate load for the system and record the performance. In order to replicate a real-world scenario we built a network of 3 instances of our system, running in disparate locations: California, Paris, Tokyo. All instances are connected to each other, and all are Ethereum verifiers. We detail the results of several scenarios, each of which consists of requests sent from the host system (the server) to its locally run DAC container. These requests result in communication between the DAC container and the locally run BP container, to performance on-chain operations which are propagated throughout the network. Each test is run for 100 seconds with requests to the system sent in parallel; 10 at a time with 2 second intervals. Each request waits to receive the appropriate response from the DAC container, we record how long this takes. We run 5 tests in total, these tests replicate the kind of operations that would be carried out by banks utilising the system. They are detailed in table 2 in order of number of blockchain transactions. Table 1 shows the number of times each test was completed during the 100 second run-time (iterations), and

Table 1: Iterations and success rates

Test	Iterations completed		Success rate (%)	
	Cali.	Paris	Cali.	Paris
Add File	151	152	99	98
Grant File Access	92	100	98	98
Grant Tier Access	86	81	98	97
Change Tier Access	66	69	99	98
Remove File Access	60	32	99	99

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

the success rate for each test. Success rate is indicated by the receipt of the proper HTTP response from the DAC container, which is only sent upon completion of the associated blockchain operations and upon receipt of the appropriate message from the blockchain container to the DAC container. Tests run from both California and Paris show comparable speed and success rates, suggesting that our system is capable of on-boarding and/or performing operations for around 10 customers every 2 seconds. Moreover, the system is on average capable of the full cycle of adding a customer, granting and checking access, and then removing access and checking again between once every 1.7 second and once every 3.3 seconds.

As can be expected local read-only operations on chain-data are much faster than full blockchain transactions. Ethereum automatically makes a local query for calls to any functions that do not affect state. For us they are calls to any 'check' function. In our 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 3698ms. Here we see the difference between the time to achieve consensus between internationally located instances, and to access a local copy of the chain, 3659ms. However, in both cases the time is small enough to ensure that the system is acceptable 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 adding of 150 new customers every 100 seconds.

6 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. It makes use of various technologies and cryptographic techniques to ensure it is secure and compliant with relevant laws. 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.

7 FUTURE WORK

Our future work will consist of an in-depth look at how to price the data shared through the system such that participating banks are remunerated for the work that they have done collecting or validating KYC documents. We aim to run larger scale tests with a larger network, with requests coming from multiple instances at the same time.

REFERENCES

[1] ABBL: Mutualisation of functions in the luxembourg financial services section (2019)
[2] Docker: Empowering app development for developers. "https://www.docker.com/" (2020), [Online; accessed 13th Jan 2020]
[3] go ethereum: clique. "https://godoc.org/github.com/ethereum/go-ethereum/consensus/clique" (2019), [Online; accessed 9th Jan 2020]
[4] Hyperledger: Hyperledger fabric. "https://www.hyperledger.org/projects/fabric" (2020), [Online; accessed 13th Jan 2020]
[5] Hyperledger: Hyperledger technology projects. "https://www.hyperledger.org/projects" (2020), [Online; accessed 13th Jan 2020]
[6] Hyperledger: Hyperledger/fabric. "https://github.com/hyperledger/fabric" (2020), [Online; accessed 13th Jan 2020]
[7] 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)
[8] 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)
[9] 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)
[10] Moyano, J.P., Ross, O.: Kyc optimization using distributed ledger technology. Business & Information Systems Engineering **59**(6), 411–423 (2017)
[11] Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
[12] 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)
[13] 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)
[14] Wiki, B.: Script. "https://en.bitcoin.it/wiki/Script" (2019), [Online; accessed 9th Jan 2020]
[15] Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper **151** (2014)
[16] 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)
[17] 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)

8 APPENDICES

```
Paris Remove File Test Results:

    add customer is true
    add file is true
      98%        68 /    1
    grant tier access is true
      97%        65 /    2
    grant file access is true
    check file access is true
    remove file access is true
      98%        61 /    1
    grant file access is false

ACheckFileAccess(ms).......: avg=141.717255 min=7.600484 med=39.319457 max=2026.793933 p(90)=265.447351 p(95)=638.374597
ACustomerAdd(ms)...........: avg=1546.122757 min=208.060549 med=1186.183344 max=6831.346925 p(90)=2594.204955 p(95)
     =4619.824519
AFileAdd(ms)...............: avg=1770.610593 min=209.87393 med=1405.430816 max=6887.593774 p(90)=3023.719372 p(95)
     =5324.099745
AGrantFileAccess(ms).......: avg=1383.022046 min=218.249344 med=1304.733651 max=2520.831624 p(90)=2481.291883 p(95)
     =2494.026936
AGrantTierAccess(ms).......: avg=1506.065996 min=237.587196 med=1502.486748 max=6994.357308 p(90)=2519.674935 p(95)
     =2539.880136
ARecheckFileAccess(ms).....: avg=60.675707 min=6.974986 med=24.733254 max=427.292495 p(90)=218.789929 p(95)=229.723793
ARemoveFileAccess(ms)......: avg=1191.159119 min=206.241319 med=1207.60996 max=2446.913211 p(90)=2040.501959 p(95)
     =2045.331459
checks.....................: 99.12%  451    4
data_received..............: 868 kB 8.7 kB/s
data_sent..................: 247 kB 2.5 kB/s
http_req_blocked...........: avg=39.75s   min=4.25 s     med=6.07 s      max=5.08ms    p(90)=10.44 s     p(95)=28.11 s
http_req_connecting........: avg=15.19s   min=0s       med=0s         max=2.7ms     p(90)=0s         p(95)=0s
http_req_duration..........: avg=1.62s   min=6.97ms    med=979.53ms   max=1m0s      p(90)=2.5s      p(95)=3.01s
http_req_receiving.........: avg=76.21s   min=0s       med=75.69 s     max=298.97 s    p(90)=108.26 s    p(95)=116.5 s
http_req_sending...........: avg=40.16s   min=13.46 s    med=23.19 s     max=3.36ms    p(90)=44.63 s     p(95)=56.6 s
http_req_tls_handshaking...: avg=0s      min=0s       med=0s         max=0s        p(90)=0s         p(95)=0s
http_req_waiting...........: avg=1.62s   min=6.85ms    med=979.4ms    max=1m0s      p(90)=2.5s      p(95)=3.01s
http_reqs..................: 455 4.549994/s
iteration_duration.........: avg=13.33s  min=5.03s     med=8.52s      max=1m7s      p(90)=14.79s     p(95)=1m0s
iterations.................: 60  0.599999/s
vus........................: 10  min=10 max=10
vus_max....................: 10  min=10 max=10
```

```
California Remove File Test Results:

       add customer is true
       add file is true
         98%        68 /    1
       grant tier access is true
         97%        65 /    2
       grant file access is true
       check file access is true
       remove file access is true
         98%        61 /    1
       grant file access is false

   ACheckFileAccess(ms).......: avg=141.717255 min=7.600484 med=39.319457 max=2026.793933 p(90)=265.447351 p(95)=638.374597
   ACustomerAdd(ms)...........: avg=1546.122757 min=208.060549 med=1186.183344 max=6831.346925 p(90)=2594.204955 p(95)
        =4619.824519
   AFileAdd(ms)...............: avg=1770.610593 min=209.87393 med=1405.430816 max=6887.593774 p(90)=3023.719372 p(95)
        =5324.099745
   AGrantFileAccess(ms).......: avg=1383.022046 min=218.249344 med=1304.733651 max=2520.831624 p(90)=2481.291883 p(95)
        =2494.026936
   AGrantTierAccess(ms).......: avg=1506.065996 min=237.587196 med=1502.486748 max=6994.357308 p(90)=2519.674935 p(95)
        =2539.880136
   ARecheckFileAccess(ms).....: avg=60.675707 min=6.974986 med=24.733254 max=427.292495 p(90)=218.789929 p(95)=229.723793
   ARemoveFileAccess(ms)......: avg=1191.159119 min=206.241319 med=1207.60996 max=2446.913211 p(90)=2040.501959 p(95)
        =2045.331459
   checks.....................: 99.12%  451    4
   data_received..............: 868 kB 8.7 kB/s
   data_sent..................: 247 kB 2.5 kB/s
   http_req_blocked...........: avg=39.75s   min=4.25 s     med=6.07 s      max=5.08ms     p(90)=10.44 s     p(95)=28.11 s
   http_req_connecting........: avg=15.19s   min=0s        med=0s         max=2.7ms      p(90)=0s         p(95)=0s
   http_req_duration..........: avg=1.62s   min=6.97ms    med=979.53ms   max=1m0s       p(90)=2.5s       p(95)=3.01s
   http_req_receiving.........: avg=76.21s   min=0s        med=75.69 s     max=298.97 s    p(90)=108.26 s    p(95)=116.5 s
   http_req_sending...........: avg=40.16s   min=13.46 s    med=23.19 s     max=3.36ms     p(90)=44.63 s     p(95)=56.6 s
   http_req_tls_handshaking...: avg=0s      min=0s        med=0s         max=0s         p(90)=0s         p(95)=0s
   http_req_waiting...........: avg=1.62s   min=6.85ms    med=979.4ms    max=1m0s       p(90)=2.5s       p(95)=3.01s
   http_reqs..................: 455 4.549994/s
   iteration_duration.........: avg=13.33s  min=5.03s     med=8.52s      max=1m7s       p(90)=14.79s     p(95)=1m0s
   iterations.................: 60  0.599999/s
   vus........................: 10  min=10 max=10
   vus_max....................: 10  min=10 max=10
```