MTChain: Identity and Confidentiality for Blockchain based Systems

Jorge Guilherme Vieira Alves
Instituto Superior Técnico, Universidade de Lisboa
jorgevalves@tecnico.ulisboa.pt

Abstract—Over the years, blockchain systems have been developing, reaching two different categories: permissioned and permissionless. Used in different contexts, each has its qualities. The first is more related to an anonymity context, while the second fits better a company, closed environment.

The work reported in this document demonstrates the implementation of a platform - MTChain, that uses a permissioned blockchain. It aims on providing identity and privacy to users, writing the necessary records on the blockchain. The platform is applied by a created prototype that emulates a notary service.

I. INTRODUCTION

Blockchains date from 2008, when an anonymous group known as Satoshi Nakamoto developed it to be the backbone of the cryptocurrency Bitcoin [1]. Its implementation was done as a part of the Bitcoin infrastructure, consisting of connected blocks of data through hashes with security as the main goal.

Early developments on the blockchain allowed for technological advances. An example of this is the Ethereum's blockchain [2] which gave different relevance to smart contracts: an automated, autonomous way of making transactions safely [3]. From this point on, some people started looking at the blockchain from a different perspective, in which blockchains, although public, could have other uses and purposes. This lead to the development of new blockchains that are more restrict and controlled, called permissioned, where not everyone is allowed to join the network and the working nodes are trusted in advance [4].

Where blockchains are safe and verifiable, they lack uses in terms of privacy, confidentiality and identity recognition. Anonymity is a priority to most of the developed system, which makes sense in the context, but only a few focus these three qualities.

Reasoning about these problems seems to lead to logical conclusions that they have become interconnected. When having strong identity, there is a need for user privacy, resorting to confidentiality upon the data that it stores. The goal depicted in this document is to create a solution that is blockchain-based, losing third party verifiers, but that also incorporates strong identity proofs and confidentiality on the system where only permissioned users are able to read the data.

This infrastructure will be sitting on top of a blockchain with public records that allow the system to be trusted, much like any other generic blockchain. The blockchain will be managed be an identity provider and/or manager, providing identity to the participants, by adding or removing users from the system.

Identity will be provided in terms of a public key infrastructure, for each participating part will have its own certificate that is known widely.

Furthermore, the values (i.e., data) involved in exchanges between users will be stored in the system, but only after going through cryptographic mechanisms, for privacy purposes, while still being verifiable in the blockchain, which represents the public part of the data.

For a better understanding on the approaches architecture, this will be explained in detail and exemplified with a use-case, in an attempt to clarify certain needs of the system by applying them to a service.

This solution is to be developed along side with the company Multicert that special-izes in the issuing of certificates, being the first to be recognized by the Portuguese National Security Cabinet as a certification authority [5] [6]. Multicert will be co-orienting the proposition over an internship. The final product should be tested in a professional, real life environment.

II. BACKGROUND AND RELATED WORK

Blockchains developed in a direction that seems to satisfy a large set of business applications. Its first implementation, behind the first version of bitcoin [1] came in 2008, and revolutionized the concept of online transaction anonymity: an infrastructure that would allow users to perform monetary transactions and remain anonymous, with no third party entity verifying their identities, but keeping a set of assurances that would make the system trustworthy.

This implementation represents the base for all blockchains. It consists of records stored in a chain. Each element of the chain is a block, where each contains transactions and metadata for verifiability and identification. The mechanisms that make this ledger consistent is the fact that the blocks are connected with each other through hashes created upon block construction. This hash is computed using the transactions hashes, block information and, most importantly, the hash of the previous block and a computed nonce. A representation of a chain of blocks can be seen in figure 1.

Computing a block hash is a complex and expensive process. The network that constructs a blockchain is a peer-to-peer aggregate of nodes that work together to maintain the ledger, but with a competitive side, which rewards monetarily the node that first reaches the desired conditions of the block hash, where the nonce comes in place. The objective is to
satisfy a certain requirement of the hash by increasing the nonce value, where this condition can be, for example, that the hash value starts with three zeros. In terms of computation, this is an expensive process, taking a lot of resources and time to nodes that compete to find it first. However, verifying this value is trivial, and modifying it would mean that the attacker would have to modify the whole chain before the attacked block as well. This leads to a robust structure, where attacking it is infeasible.

Bitcoin was the first system to use blockchain to keep transaction records. Besides, it created a stack-based language that runs by script for transactions automation. However, this implementation was somewhat weak, failing to provide programming paradigms necessary for the execution of complex operations. This happen due to the nature of the system, where Ethereum [2] decided to take a step further and create what is known today as smart contracts.

Ethereum is, at this moment, the second biggest cryptocurrency in the market. When created, it innovated on the concept of smart contracts, creating limits of execution and security mechanisms to improve them. These contracts run transactions automatically, which is an application desired feature [3].

Where some saw opportunities to leverage the anonymity and the potential of the system in a public way, others saw fit in closed environments, more restrict and conditioned. These separation became what is known as permissionless and permissioned blockchain systems. The first, a public network: records are public, anyone can participate. The latest, a private network: records are public, but require authorization, and so does the use of the network.

The biggest example of a permissioned blockchain system is the implementation provided by the Hyperledger collaborative association of a blockchain network: Hyperledger Fabric [7]. Composed by 3 types of nodes: orderer, peers and CA; this ledger network works by gathering peers into organizations, performing a plugable consensus protocol on the orderers and providing them with identity and the means for TLS, this last done by the CA node.

Through an API, implemented using SDKs, applications connect to the blockchain network, thought the peers, issuing requests that trigger operations on the smart contracts, named chaincode in this context. These operations are executed and usually lead to either an update on the blockchain (a transaction) or a query.

To provide identity to entities, other blockchain implementations were created. For example, namecoin [8] was a system developed to provide a decentralized naming service, creating a stronger, more resilient DNS provider, to counter the actions of governments and attackers that intent on shutdown certain services/websites. From this, came blockstack [9]. Blockstack works under the same principle, creating decentralized names for its users, slightly breaking the anonymity barrier created by bitcoin. By attaching public keys to names, these are later identifiable and verifiable publicly, since this is based on bitcoin’s blockchain.

The strategy used by blockstack incorporates the use of a PKI [10], an identification system that uses public and private keys to recognize owners. These diverge into certificates, which is a fairly used concept all over the internet. Certificates contain the public key of the owner, as well as some information regarding it. Associated is the private key, which is known only by the owned. Together, these provide several cryptographic mechanisms: digital signatures, where the signer uses its private key to create proof that it was he created the signature of the message, bringing integrity and non-repudiation to the system; asymmetric encryption, where a message sender can encrypt the message with the public key of the user and this can decrypt it by using its private key; authentication; among others.

To provide keys, a certified entity has to participate. This entity is what is called a CA, with issues certificates to the desired clients, assuring that they come from a reliable place. An example of a Portuguese CA is Multicert [6] [5], the first CA in Portugal to be recognized by the National Security Cabinet, and the supporter of the development of the work in this paper.

On the other hand, privacy and confidentiality is an issue to be considered. Some systems develop means to provide these qualities, ensuring that the client’s private data is safe. Enigma [11], for instance, provides a platform on top of a blockchain that, through the use of Multi Party Computation, ensures that no node ever get the complete information that the user sends. Besides that, smart contracts in this context are divided into two parts, a public part and a private part. These separated execute the smart contract correctly, but the nodes do not have access to understand what is done.

Hawk [12], in the same topic, takes an approach to smart contracts. With the use of a specific processor and a finely-trusted third party, it provides a framework for the development of smart-contracts that are concerned about transaction privacy. The goal is to create mechanisms for contract execution without the need to know the value to be processed before-hand.

III. IMPLEMENTATION

MITChain implementation revolves around adaptability to the use of the blockchain. In simple terms, it contracts a middle connection between client applications and the ledger. Its purpose is to create the means for the communication with the blockchain network, but safely: only after several
nodes with different tasks that work together: Peers provided by Hyperledger Fabric’s blockchain service: a set of the applications layer and the blockchain layer. The base is before, the platform - MTChain, stays in the middle, between blockchain usage.

Decomposing the layered architecture, we see that, as stated before, the platform - MTChain, stays in the middle, between the applications layer and the blockchain layer. The base is provided by Hyperledger Fabric’s blockchain service: a set of nodes with different tasks that work together: Peers, which create and host the blockchain, working in a peer-to-peer protocol, receive the requests for chain updates and queries, running the provided smart-contracts code; Orderers, with the responsibility of managing the blockchain entrances logistics (temporal consistency, for instance), running a consensus protocol; Fabric-CA, a network identification analyzer, authenticating and identifying the network nodes. Together, these nodes form the infrastructure that manages the blockchain, creating and inserting new blocks in the chain. When receiving transactions, the peers evaluate them, by testing it against their smart-contract code. If validated, then the transaction is sent to the orderers, which will organize the transactions timeline. After a pre-defined number of transactions is received, a block is created in the peers and introduced to the blockchain.

This process involves an extra entity, a Hyperledger Fabric’s client. MTChain takes this place in the system. By using Hyperledger Fabric’s Java SDK [13], communication is facilitated, providing methods for smart-contracts management and blockchain updates/query. To do so, the SDK provides methods to connect to a channel on the network and different types of users, with different permissions.

MTChain, when connected, does not request knowledge on the chain. Instead, it keeps a persistent record stored in the Database. This component contains application related records, being these crucial to its correct functioning. However, the main focus that we are addressing is the storage of blockchain information: in the database, MTChain stores verifications upon data and authentication can this be sent to the blockchain.

As seen in Figure 2, MTChain is divided into services that work together to accomplish a secure and effective network. Located on top of the Hyperledger Fabric network, it provides authentication mechanisms for applications and an abstraction to blockchain operations, both for administrating purposes and blockchain usage.

![Fig. 2: The layered view of MTChain](image)

Creating users and installing smart-contracts into the blockchain is done in the Backoffice, an application with the function of managing the system and the blockchain. In this application, the network administrator can see information regarding the existing applications, users, nodes in the network and smart-contracts installed. It can also upgrade/downgrade/install smart-contracts: upgrade the version of the smart contract, downgrade the version to provide alternatives to errors on new versions, or install a new smartcontract. Regarding users, the administrator has the function of adding new users, giving them a name, an application to be associated with, and a certificate. These users can be deleted and modified at any time, which is considerably important for renovation of certificates.

Finally, the applications layer in the figure represents the client applications. This is a dynamic module, where there can be many applications with different functions, leveraging the flexibility of the below layer methods. Having several different applications is on of the goals of the system, where it should not be restricted to one solid use. One example of an application is the Notary application, which also works as a Proof of Concept to this system. The following section describes the prototype.

IV. PROTOTYPING

Creating a PoC for this system required the implementation of an application that would demonstrate its capabilities. The Notary application was created with the purpose of providing it, simulating a notary service online, decoupling third parties verification entities.
A notary is a trusted, recognized person that witnesses the signing of a document. The goal is to prove that the signature happened at a certain point in time and that it was performed by the individuals whose signatures are listed in the document.

Implemented on top of MTChain, the notary application aims to provide the same guarantees without extra costs or the effort of dislocation/making appointments for the users. Besides that, when working with a notary there’s a human factor, susceptible to corruption or even the slightest miss care, where the notary may see the content of the document in order to verify it. In this prototype, the system never sees the content of the signing document.

To create an application such as the one described in this section, the platform provides mechanisms to facilitate the tasks on the blockchain. MTChain contains the definition of generic methods that manage the logistics of the applications relation with the blockchain. Therefor, the developer job is to incorporate these methods with the logistics of its application, deciding what to store in the blockchain, according to the smart-contract that it will be running - this also created by the developer.

Bringing flexibility for the creation of the application also brings certain responsibilities. As in the notary application, some information should not be seen by the servers running the service. This creates a sense of conscience on the developers side, where the person behind the development of the applications should also consider ways to conceal data, keeping it private. For instance, on the notary application, to prevent disclosure of information to the server, the document is never sent to the server. Instead, on the client side, the document is uploaded and its hash computed, being this the one sent to the server side.

In this application, users are authenticated by certificate. The server requests the browser for a certificate that is checked against the user related information, its certificate more specifically. This attributes a session id to the user in question. Another viable option for authentication (which was also considered and might be implemented in the future along the already existing authentication method to provide another layer of security) is username/password authentication. To protect the user, the password would not be transmitted, but its hash. This would be stored in the database along with salt that would be hashed a pre-defined number of times. The hash and salt would be kept in the database, where the salt would always be the same value. And so would the iterator of the number of times that the password is hashed, being this kept in a configuration file.

Signing the document also differs from a notary in a simple way: each signature endorses the previous signature. Due to the logic of the application, it was decided that there should be a queue of users to sign with a specific order. However, this is not a mandatory feature, but simple business logic. The process of signing a document is constrained to a temporal pool, where several user operations occur that change the state and current signer of the document.

The first operation would be the upload of a document to the system. A user enters a document name and description (making it identifiable by other signing users), the application users that are meant to sign besides the uploader, and the document to be signed. This submission triggers on the system the start of a signing operation, which will lead to the platform storing the hash of the document, along with its signature, on the blockchain, starting the pool for the signature, creating a record on the upload time of the document.

Following this operation, each one of the users will found the document in its personal page. At their turn, they will be prompted to sign it by providing to the client their private key and the password. This is used by the browser to create the signature on the document, by signing the previous signature on the document, creating a chain. Each one of the signatures is sent to the network as a transaction to be later stored in the blockchain.

All the selected users must sign the document to initialize its completion. After the last user signs it, the server is triggered to start the finalizing operation, which closes the temporal pool, concluding the time line of the document signing. The server signs the final signature, storing it on the blockchain.

The page contains also a history section, where users can see the documents they have or will be participating in. This sections shows a state of the document and leads to a graphic time line demonstration, that contains the hash of the document, the signatures of the users that already signed and a current state of the document (finished or not finished).

The following sub section describes the flow of execution of a signature process with an example.

A. The flow of a signature process

Let us consider a simple document to be digitally signed by two different users, Jacob (which will upload the document) and Maria. For simplicity, we will consider that the both users are already present in the network and authenticated.

Jacob enters the website. It moves to the “Start New Signature” section of the page, finding a form to provide the document and information regarding it. It selects Maria as a signer and sends the form to the server. As state before, the server will receive the hash of the document only, SHA1, which will sign. Afterwards, it sends it as a transaction to the blockchain network, starting a signing process of the said document. On the blockchain, it stores the document name, hash, its signature and a timestamp.

After the server is done with the previous operation, the page refreshes. Jacob can now find the document in the “Documents to sign” section, while Maria can only find it in the “History” section with the indication that Jacob is signing. By pressing the document name, the page loads a modal form, which asks for the users signing key (the one associated with the certificate provided for authentication) and its password. Using the SHA1withRSA algorithm, the client generates a signature over the signature performed by the server when it started the process. This signature is sent to the server in a hexadecimal format, which the server converts back to binary to verify it against the user’s certificate. If it verifies,
the server sends the signer name, document name, signature and timestamp to the blockchain to persist the information. Afterwards, the same process repeats with Maria, with the exception that it signs over Jacob’s signature.

Finally, after Maria’s signature is stored in the blockchain, MTChain verifies that she was the last user on the queue, meaning the document is finalized. The platform gets Maria’s signature, signs over it and sends it to the blockchain, indicating that the signing process is finished and successful. In the end, the document signature will be as depicted in figure 3.

V. Evaluation

This chapter describes the evaluation done to the system, MTChain. To perform this evaluation, several tests are run using the implemented prototype, the Notary application.

The goals of MTChain are split into two main subjects: security and performance. This being, the evaluation was designed to answer the following questions:

1) Does the system, allied with the prototype, allow the necessary conditions for user privacy, identification and overall security?
2) Since interoperability is a constant topic in this document, does it provide acceptable performance measures?

The following sections address these two questions, starting with the security and identity concerns, moving to both latency and throughput measures.

A. Security and Identity

When considering security measures to protect both the system and its users, the best approach to take is to use already defined/implemented methods that are assured by trusted entities.

This system is no exception: security tools were not custom made, but instead reused, where security is assured by the usage of already existing methods and objects.

1) Privacy: In this specific system, privacy is the most important feature in terms of security. Losing privacy might mean that a private, sensitive document is leaked to the outside or to the platform itself, which is undesired. To ensure this will not happen, both the platform and the applications’ developers work together, where both apply logics to prevent this from happening.

On the server side, the platform is only ready to deal with documents’ hashes and public certificates. This ensures that the documents signed and uploaded to the client never leave the client, not even to share with other users permitted to view them. This is allowed, for logically every user should have the document which they are going to upload/sign.

On the client side, to avoid the transmission of the document’s sensitive information, when a user uploads a document and presses the upload button, it does not trigger the POST from the form. Instead, it triggers a JavaScript function that will retrieve the document from the form. This document will be encoded and hashed with the SHA1 algorithm, which is then sent to the server in a hexadecimal format.

With these two considerations, the privacy of the user’s documents is not violated. Allied to that, the server keeps only the public certificate of the user, which does not disclose the user’s private key, but used to verify the necessary information.

2) Confidentiality: Although the information passed through the channels is never too sensitive and often used to publicly verify the user’s interactions, this is still transmitted through an SSL secured channel.

The client side is also authenticated, where the server requires also the client’s certificate in order to accept communications with it.

3) Authorization: Authorization over services requires authentication of the users. This mechanism is to be done as the applications programmer choose, but the standard is the use of the users certificate, whose CA is located in the server keystore and the certificate itself generated by the browser from the .p12 file that contains the private key. This is later compared with the system owned certificates. If it returns true, then the user can be authenticated. To improve this mechanism, the use of a username/password is also possible, where this would preferably be kept in the server after being hashed with a salt value an n number of times. n which would be kept in a configuration file.

4) Repudiation: All communications and interactions with data are done with users that are authenticated in the system. This authentication is associated to their certificate. Besides that, the signing operations are verified against the user’s certificates.

5) Database data breaches: The database accesses and privileges are different depending on the type of user that accesses it. Also, username/password based authentication is required to access the data.

B. Performance

Performance over a system that comprises web applications is a fundamental concern. Therefor, the need to reach a certain
acceptance level is crucial and must be tested.

The most regular tests on performance are latency and throughput, which are exactly the tests that will be performed, with different variants. These tests were performed for both the Notary application and the Admin application.

The environment for these tests is done using:
- A single instantiation of the admin application;
- A hyperledger organization containing one ordering node and two peers.
- A single instantiation of the Notary application;

All the values found in tables result from a media of thirty measurements. The tests were run on a localhost machine, a Toshiba laptop with 16Gb of RAM and an i5 Intel CPU. The whole system ran in this machine, disregarding the internet latency from the measures. Tests were automated, ran by calls and specific waiting times, discarding users delays in interactions.

C. Performance over the Admin application

Understanding the administrator application, the operations it performs are simpler but require more time to be finished than the ones in users applications. The processing is mostly located on the Hyperledger Fabric, creating a waiting period on the MTChain administrator, a bottleneck in Latency. This is explained in depth in the following section, which evaluates the administrator application in terms of latency

1) Latency: On the administrator side, it would only make sense to test only latency, once throughput can not be affected by concurrency or other variants. So, in this section, we take a look over the operations that the system performs on the administrator side: the installing and instantiating of smart-contract, and the upgrading and instantiating of the smart-contract. These operations are tested along with the size of the files being installed, which provide the comparing factor amongst different times.

On table 4 we can observe the first tests performed upon the system: Latency resulting from the installations of smart-contracts with different file sizes. Each one of the left names represent a different smart-contract file, each with different complexities, increasing in an ascending order across the table. Installing smart-contract involves an install and an instantiate operation transmitted to the network.

These times are measured in order to evaluate the relation between the smart-contract installed, the operations they execute and the file size. Installing smart-contract involves an install and an instantiate operation.

D. Performance over the notary application

From the resulting time, we can verify that the size of the smart-contract is not a relevant issue when installing. This is due to the nature of the network, where each peer compiles the code and communicates to the orderer that they did so.

Next, we evaluate the latency in upgrade operations. In table 5, we take the same approach as before:

Once again, the times do not change a lot according to the size of the file, where we can deduct that the size of the file is not a constraint on the time of execution of smart-contract installations and upgrades. Downgrades were not considered, since the operation consists on an immediate action on the variables on MTChain.

In resume, each operation took between 40 to 50 seconds to execute a change in the smart-contract. This is a good time, if we consider that this means a downtime of around 1 minute per upgrade on applications. Choosing less used moments of the application to do so is an effective measure to upgrade without affecting user experience.

In resume, each operation took around 45 to 49 seconds to execute a change in the smart-contracts. This is a good time, if we consider that this means a downtime of around 1 minute per upgrade on applications. Choosing less used moments of the application to do so is an effective measure to upgrade without affecting user experience. The following section addressing the testing done to the notary application.

Fig. 4: Time relation of smart-contract installation related to size of file

Fig. 5: Time relation of smart-contract upgrades related to size of file

<table>
<thead>
<tr>
<th>Chaincode</th>
<th>Size</th>
<th>Time</th>
<th>StandardDeviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>cc</td>
<td>2.8kB</td>
<td>49s</td>
<td>3.56s</td>
</tr>
<tr>
<td>fabcar</td>
<td>6.4kB</td>
<td>47s</td>
<td>3.43s</td>
</tr>
<tr>
<td>marbles</td>
<td>24.9kB</td>
<td>47s</td>
<td>4.14s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chaincode</th>
<th>Size</th>
<th>Time</th>
<th>StandardDeviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>cc</td>
<td>2.8kB</td>
<td>47s</td>
<td>3.17s</td>
</tr>
<tr>
<td>fabcar</td>
<td>6.4kB</td>
<td>45s</td>
<td>3.27s</td>
</tr>
<tr>
<td>marbles</td>
<td>24.9kB</td>
<td>47s</td>
<td>3.14s</td>
</tr>
</tbody>
</table>
by the platform on the server side. All these operations involve queries or updates on the blockchain.

All the times are reasonable, but as the number of documents increase, so does the time to load their information into the timeline. This is due to the fact that the filling of this section is done by querying the blockchain, which is a light process, but becomes heavier as the number of queries goes. The suggestion to improve this is to query documents individually and upon click.

To the reads consideration, all the times measured in Start signature and Sign document were done discarding the time of loading of the timeline, for a more precise approach.

The complete execution time represents the time elapsed in:

1) Loading the index page;
2) Starting a new signature (which involves server signing and blockchain update);
3) Loading the index page;
4) Signing the document (which involves client signing and blockchain update);
5) Finishing the document (on the server side, by signing the hash one last time);
6) Loading the index page;

The complete execution was done with a single user only, and it considers the times to reach the parts of the page. Each index loading includes the query of the document, except for the one in point 1, where the blockchain would be empty and had nothing to load.

2) Throughput: To test the throughput of the system, several requests were made over a period of time, where the frequency of these increase over each minute. The requests are done by one user only, representing 3 signatures, one over the browser, two on the server side. These are done by loading the index page, starting a new signature and signing the document: a complete process.

Over the course of 10 minutes, 175 requests were done with an increasing load over each minute, as seen in table 7. As seen in the previous section, each request takes around 1300 milliseconds to complete, where the expectations to these tests is that the system works regularly until minute 8 - 9. By minute 9 - 10, the system should start delaying requests, but without losing any.

<table>
<thead>
<tr>
<th>Interaction</th>
<th>Documents in History</th>
<th>Type</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load index</td>
<td>0</td>
<td>GET</td>
<td>151ms</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>GET</td>
<td>162ms</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>GET</td>
<td>262ms</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>GET</td>
<td>646ms</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>GET</td>
<td>1897ms</td>
</tr>
<tr>
<td>Start signature</td>
<td>-</td>
<td>POST</td>
<td>582ms</td>
</tr>
<tr>
<td>Sign Document</td>
<td>-</td>
<td>POST</td>
<td>285ms</td>
</tr>
<tr>
<td>Complete Execution</td>
<td>-</td>
<td>-</td>
<td>1371ms</td>
</tr>
</tbody>
</table>

Fig. 7: Number of requests per each minute over 10 minutes, testing throughput.

<table>
<thead>
<tr>
<th>Minute</th>
<th>media time to process</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 1</td>
<td>1149ms</td>
</tr>
<tr>
<td>1 - 2</td>
<td>1244ms</td>
</tr>
<tr>
<td>2 - 3</td>
<td>1345ms</td>
</tr>
<tr>
<td>3 - 4</td>
<td>1176ms</td>
</tr>
<tr>
<td>4 - 5</td>
<td>1183ms</td>
</tr>
<tr>
<td>5 - 6</td>
<td>1143ms</td>
</tr>
<tr>
<td>6 - 7</td>
<td>1193ms</td>
</tr>
<tr>
<td>7 - 8</td>
<td>1545ms</td>
</tr>
<tr>
<td>8 - 9</td>
<td>4742ms</td>
</tr>
<tr>
<td>9 - 10</td>
<td>10104ms</td>
</tr>
</tbody>
</table>

Fig. 8: Time it takes a request to be processed in both normal and peak load execution moments.

In table 8, we can find a media of the time that took each request to finish in every minute. This tests throughput through both normal functioning and peak load moments.

As expected, the values increase in the last two minutes. However, this increase is acceptable in the matter that either 4 or 10 seconds do not represent such a big impact in the system, specially in this nature, where moments like this would rarely happen. However, this could be improved by introducing parallelism into the network, distributing the computation. A front-end server would distribute the workload to several back-end servers, increasing the number of requests necessary to reach a peak load moment, scaling the system horizontally.

E. Resume

In the end of the tests, the results are as expected. Security wise, the assurances made by the used technologies are overall satisfactory and sufficient. Performance, on the other hand, is always conditioned by the operations performed interoperability, where times can not be improved. With this constraint in mind, we think that the times are acceptable for the sake of the operations performed, leading to an user experience that is satisfactory and light for normal executions but not so much for peak load executions. However, the systems usage is not expected to reach this point, and, if it does, there are solutions to solve it. In a real environment, there would be several
machines running the service and there would be also delays coming from infrastructure logistics, such as internet delays or servers overload. In this environment, we would expect better throughput times, configuring a front-end load-balancer that would distribute to several back-end servers. However, it would be expect a higher latency due to load-balancing mechanics and the extra connections.

VI. CONCLUSION

In this document, a solution on a system that incorporates blockchains, identity management, privacy and confidentiality was described: MTChain. It interacts with a Hyperledger Fabric network, a component that constructs and manages the blockchain. The platform goal is to provide the necessary means for applications to run on top of it, while not having to worry about method definition regarding communication with the blockchain. To create a more visible concept of this work, an application was developed - The Notary. This works as a notary service, performing as a witness to a document signing.

The system developed achieves a first approach to identity management, privacy and confidentiality on a service that is today done by an entity certified, providing even improvements to it, such as the fact that no one but the signers see the information of the document.

When evaluating the system, the results showed that it takes an average of 1.5 seconds to sign a document. Comparing to a physical dislocation to a notary office, this is a huge saving point. In technical terms, 1.5 seconds is also a good time for the user to perform it, taking into account the operations performed between that fraction of time.

Future work for this project seems endless at a first sight, when realizing the potential of the blockchain and the platform itself. However, some priorities should be considered when thinking about what would be done to improve the technology and the business logic.

Firstly, Hyperledger’s network uses an identification system based on X509 certificates, which authenticate both the nodes of the network and the clients of the service. These certificates are self-signed at the moment, generated upon network creation by a tool provided by Hyperledger. In the future, these certificates could be replaced by Multicert’s certificates, a step further in the certification of the system. This would be useful for the three parties involved: Hyperledger, which would benefit from a different implementation example of certification amongst nodes; Multicert, which would see another fit to its CA; the system described in this work, which would improve the authentication mechanisms between the client and the trust on the network.

Another feature to be implemented would be the discarding of the database server, making the platform a distributed, peer-to-peer network. This would increase the complexity of the network largely, but create a distributed hash-table and a storage facility distributed, improving qualities such as user usage and availability. On top of this improvement comes the possibility for the storage of the documents themselves with cryptography properties, allowing the uploaders to later view their documents, but restricting it from the peers. This would require strong synchronization mechanisms and a network of peers to be working, so it should be developed in an advanced state of the platform, where it is justified.

REFERENCES