UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE INFORMÁTICA

A Know Your Customer solution over the Portuguese Citizenship Card

Mestrado em Engenharia Informática
Especialização em Arquitetura, Sistemas e Redes de Computadores

Hermínio Miguel Sobral Tavares

Trabalho de Projeto orientado por:
Professor Doutor Carlos Coutinho
Professora Doutora Maria Isabel Nunes

2019

Agradecimentos

Em primeiro lugar, queria agradecer à minha **família**, que sempre esteve disponível em tudo o que precisei. Sempre me apoiaram incondicionalmente em todos os momentos da vida.

Queria deixar também o especial agradecimento a ambos os professores orientadores que me acompanharam neste trabalho, pois foi também graças a eles que consegui atingir este marco. Em especial, ao **Professor Carlos Coutinho** por me ter aceite como seu orientando; de outra forma, provavelmente não teria concluído o mestrado este ano letivo.

Devo também um especial agradecimento à **Caixa Mágica Software**, que me deu a possibilidade de frequentar as aulas durante o período laboral; infelizmente, nem todas as empresas concedem este tipo de benefícios aos seus colaboradores. Para além disso, não me posso esquecer do apoio que tive de todos os meus **colegas**, que sempre se mostraram disponíveis e compreensivos para me ajudar no que fosse necessário.

Um agradecimento especial ao **Bruno Cipriano**, que foi meu professor durante a licenciatura e que hoje é um grande amigo. Devo a ele grande parte dos conhecimentos que tenho hoje em dia. Mesmo fora da vida académica e das suas responsabilidades como professor, sempre se mostrou disponível para me ajudar em diversas situações que careciam de conhecimento que eu não possuía.

Finalmente, mas não menos importante, queria também agradecer ao **Professor Pedro Alves** que também me acompanhou durante a licenciatura. Também sempre se mostrou disponível para me ajudar e foi um dos impulsionadores que me levou a fazer o mestrado.

Dedicatória

Para o meu pai, o meu lutador.

Resumo

Nos dias que correm, temos observado avanços significativos nas diversas áreas da informática. Não obstante, ainda é frequente encontrarmos, nas soluções existentes no mercado, aquelas que exigem processos manuais e por vezes morosos, como por exemplo, os relacionados com a atividade de adesão, engajamento, ou, usando o termo técnico, *Know Your Customer* (KYC). Estes processos consistem em identificar e validar clientes ou membros de uma instituição. Se um cidadão se deslocar a um banco para abrir uma conta ou pedir um empréstimo, é obrigado a entregar um conjunto de documentos e provas da sua identidade, documentação essa que será alvo de análise humana. Em média, de acordo com estudos efetuados, os atuais mecanismos de KYC consomem às grandes empresas e instituições (com cerca de 10 mil milhões de dólares de lucro anuais) perto de 150 milhões de dólares. Para além disso, em média, o custo de um processo que envolva mecanismos de KYC nos trâmites atuais, varia entre os 15 e os 20 dólares.

Adicionalmente, dado que estas tarefas são realizadas manualmente por olho humano, estão – como seria de esperar – sujeitas a erros, assim como a um maior consumo de tempo em relação ao comportamento de tarefas automatizadas. Hoje em dia num mercado globalizado e extremamente competitivo, é fulcral que estas empresas e instituições sejam o mais eficientes possível. Neste caso concreto, o facto destas tarefas de KYC estarem a ser realizadas manualmente, prejudica em muito a eficiência destas entidades, uma vez que poderiam realocar este esforço noutros cenários que lhes poderiam ser mais vantajosas.

De forma a agilizar estes processos, a Caixa Mágica Software serviu de incubadora para um projeto denominado WalliD. O WalliD é um protocolo *open source* que tem como objetivo guardar identidades na *blockchain* Ethereum. Embora já existam algumas soluções deste género, este protocolo garante também a segurança e a confiança destas identidades, desde que as mesmas tenham associado um certificado X.509 confiável. Embora esta solução esteja assente numa *blockchain*, a identidade do cidadão é cifrada com recurso a criptografia assimétrica, possibilitando que apenas o próprio tenha acesso aos dados. Desta forma, o cidadão é livre de deliberar a quem facultará os seus dados pessoais para efeitos de verificação da sua identidade.

A utilidade deste protocolo é notória, especialmente porque permite de forma fácil e transparente a implementação de um sistema de KYC automatizado, eliminando assim o erro humano e automatizando processos que anteriormente eram manuais.

Este trabalho retrata o desenvolvimento do primeiro KYC assente no protocolo WalliD utilizando o Cartão de Cidadão português. Este sistema de KYC resulta numa arquitetura genérica e modular que possibilite, através de testes unitários, testar as funcionalidades implementadas, assim como adicionar novas identidades para além do Cartão de Cidadão português, com o mínimo de esforço possível. Para isso, foram utilizadas técnicas de injeção de dependências, uma vez que promovem a construção de módulos com uma elevada independência entre eles.

Este sistema de KYC contempla três grandes validações. Em primeiro lugar, a validação do certificado existente no Cartão de Cidadão português. Esta validação permite identificar casos em que o documento de identificação já se encontra expirado, revogado, ou que não seja confiável por impossibilidade da criação de uma cadeia de certificados de confiança. Uma vez que a identidade dos utilizadores é guardada na *blockchain*, o passo seguinte resume-se à verificação de uma assinatura criptográfica disponível pelo Cartão de Cidadão português, aplicada à *wallet address* da conta Ethereum do utilizador em questão. A verificação desta assinatura é realizada com recurso ao certificado verificado no passo anterior. Finalmente, é necessário aferir a validade dos atributos de identidade e de morada do utilizador. Esta validação é efetuada recorrendo a um ficheiro existente no Cartão de Cidadão português denominado de Document Security Object (SOD). Este ficheiro contém um conjunto de *hashes* que são gerados através da concatenação de todos os atributos de identificação e de morada do utilizador. Seguidamente, são assinados criptograficamente de forma a serem verificáveis através de uma chave pública existente num certificado digital residente no ficheiro SOD. No entanto, a verificação destes atributos só se considerará válida caso este certificado seja confiável. Para isso, é necessário verificar se o mesmo não se encontra expirado, nem revogado e se é possível construir uma cadeia de certificados de confiança.

Para além da definição da arquitetura, este trabalho também resultou numa implementação piloto para o já referido Cartão de Cidadão português. Esta solução foi realizada através da linguagem de programação Java, e a razão da escolha desta linguagem prendeu-se essencialmente por dois motivos, *(i)* é uma linguagem *open source*; *(ii)* de forma a facilitar a implementação de certas funcionalidades, recorreu-se a uma

biblioteca também *open source* denominada de Bouncy Castle. Visto que esta biblioteca apenas está disponível para C# e Java, optou-se uma vez mais por esta última. Ainda de modo a que esta solução fosse o mais modular possível, foi utilizado um padrão de injeção de dependências através de uma biblioteca desenvolvida pela Google denominada de Guice. Desta forma, foi possível construir uma solução que abstrai na totalidade o documento de identificação em questão (neste caso, o Cartão de Cidadão), garantindo assim que os módulos desenvolvidos sirvam para outros tipos de documentos. Adicionalmente, a utilização deste padrão facilitou o desenvolvimento de testes unitários.

Atendendo às necessidades deste trabalho e da linguagem de programação utilizada, optou-se por gerar uma biblioteca no formato Java Archive (JAR). Assim, um programador poderá encapsular o serviço de KYC num *web service*, através de um servidor que responde com recurso a *sockets*, entre outras formas, para que não exista uma dependência relativamente à tecnologia utilizada. De forma a facilitar a instalação, manutenção e a escalabilidade deste serviço, este *web service* foi configurado numa imagem de Docker.

Pretende-se, no final, que este trabalho dê resposta a quatro questões de investigação, nomeadamente, *(i)* se é possível verificar a identidade de uma pessoa através de uma *blockchain*; *(ii)* se com a utilização da *blockchain* é possível mitigar o uso descontrolado da nossa identidade, no sentido em que, ao facultarmos a nossa identidade para identificação de terceiros, perdemos o rasto completo do nosso documento de identificação; *(iii)* se através dos certificados digitais é possível aumentarmos a eficiência da forma como as identidades são verificadas, minimizando assim o erro humano; finalmente, (iv) se é possível, através de tarefas automatizadas, verificar identidades mais rapidamente, em contraponto com as tarefas realizadas atualmente de forma manual.

Caso seja possível dar resposta a todas estas questões, a utilidade do protocolo do WalliD pode ser demonstrada através de um caso de uso, que neste caso envolve a entidade bancária fictícia denominada Credibank. Esta empresa tem como objetivo fornecer créditos aos seus clientes e, para tal, necessita de validar as suas identidades. Através do mecanismo de KYC desenvolvido neste trabalho, o Credibank poderá atuar somente como uma entidade *online*.

Visto que o WalliD privilegia a metodologia *open source*, tanto o código da biblioteca desenvolvida, assim como, o código do *web service*, serão disponibilizados de

forma *open source*, permitindo assim a melhoria continua assim como dos processos implementados.

O trabalho descrito neste documento já conta com uma publicação científica na 9ª Conferência Internacional em Sistemas Inteligentes IEEE-TEMS realizada na Madeira, Portugal, no ano de 2018 com o título "*WalliD: Secure your ID in an Ethereum Wallet*". Adicionalmente, encontra-se em processo de aprovação um segundo artigo na revista "*IEEE Instrumentation and Measurement Magazine*", com o título "*Instrumentation and Measurement context: From a methodological point of view*".

Palavras-Chave: *Know Your Customer*, WalliD, X.509, *blockchain*, identidade.

Abstract

At the present time we observe several improvements in many informatics topics. Despite that, we still find too many manual processes which could be automated such as the Know Your Customer (KYC) processes. The goal of these processes is to identify and validate current or future customers or members of an institution. As an example, trivial operations such as opening a bank account, at least in Portugal, is a task that in most of the cases is not able to be performed online. The process must be conducted in person, involving signing and delivering all documentation needed or uploading a set of documentation that needs to be analyzed and checked. Additionally, that personal documentation is manually verified by a bank employee, and of course this process can be subject to errors, resulting in further delays to the account approval.

In order to solve these issues, Caixa Mágica Software, on its Startup Lab, developed and incubated a project called WalliD. WalliD is an open source protocol whose goal is to solve the type of issues aforementioned. This means that with the proposed solution, users are able to store their own identity in the Ethereum blockchain. Although the market already contains some solutions with similar purposes, WalliD brings trustworthiness to the users' identities as long as they have a X.509 certificate attached. Despite having the user's identity being stored in a blockchain, only the user himself has access to it because that data is encrypted using an asymmetric pair of keys. This solution gives the identity owner the ability to choose who, and when, an entity has access to his identity attributes.

As an open source protocol, anyone can help improve it and verify how secure and simple it is. Developing KYC solutions based on WalliD could avoid human misbehaviors and speed up this kind of processes as well.

This work describes the development of a KYC solution based on WalliD over the Portuguese Citizenship Card (PTCC).

Keywords: Know Your Customer, WalliD, X.509, blockchain, identity.

Contents

List of Figures

Chapter 1

Introduction

The evolution of computer science has provided new ways to improve and create new markets. However, there are still areas which have not yet benefited from those improvements. For instance, if we want to open a bank account in Portugal, in most cases we are forced to be physically present at the bank institution. This happens because we still must prove our identity by old-fashioned ways and processes; consequently, a person will receive and manually validate the received documentation such as our identity card, home address, among other information.

The goal of the project WalliD [1] is to connect the physical world to trustful digital identities in a secure, decentralized, transparent and immutable way. In order to do that, those identities are encrypted by the owner, and then stored on a blockchain through a smart contract. Hence, any citizen will be able to perform a Proof of Identity (PoI) in such a way that it will be possible to avoid the legacy process described in the previous paragraph.

Actually, the project WalliD was conceived and designed to accept any kind of trustful digital identity. The integration with the Portuguese Citizenship Card is simply the first full-featured example use-case of this process. In this case, it is possible to perform a PoI because the Portuguese Citizenship Card contains consistency and security information such as a X.509 digital certificate [2] issued by the Portuguese governmental certification authority, which makes it possible to anyone to perform proof of identity by chain of trust.

1.1 Motivation

Old-fashioned KYC mechanisms make companies expenses grow. Large financial institutions (with an average annual return of $10 Billion USD) have an average annual cost of $150 Million USD with KYC processes and the average annual cost for the same practices on other financial firms is around 60 Million USD [3]. Currently, the on-boarding costs for businesses with KYC Processes range from $15 USD to $20 USD per process and demand a renovation of due diligence validations periodically. High expenses are not the only issue of those KYCs – since they are performed through a human eye, it is only natural that some mistakes arise. From the moment the customer's personal documentation is delivered, these processes usually take a long time, too much time, and may block the opportunity for businesses to be performed.

Moreover, sometimes the customer's documentation flows through unsafe channels – for instance, it is usual that, in the process of opening a bank account, a copy of its citizenship card is produced. Hence, the customer is trusting its identity not only on the bank institution, but also on the employee. Despite of that, identity trust can be considered a well-solved problem in the physical world. Every jurisdiction has its own identity issuer organization which is responsible for certifying every citizen's identity attributes and issuing a respective proof of identity, usually represented by physical objects like citizenship cards and passports, and everybody, from citizens within those jurisdictions to organizations, recognize these certificates as trustful. Unfortunately, few identities are globally accepted. Each country may have their own way to issue and validate its citizens' identity.

In order to solve this problem, WalliD has a modular, customizable, decentralized and transparent information structure and architecture. Any trusted identities that use X.509 certificates are able to join and use this protocol. Companies are also able to develop their own KYCs through automatic mechanisms, avoiding high expenses and human misbehaviors.

1.2 Goals and Research Questions

The main goal of the work here described was the development of the first KYC based on the WalliD protocol, which uses the Portuguese citizenship card. The output of this solution is a generic architecture with an open source implementation. As a result, any entity which wishes to use a KYC based on Portuguese citizenship card can use this implementation, or even develop its own solution based on this architecture. The contributions of this work for WalliD can be summarized as:

- Develop a generic architecture for KYCs systems that will use WalliD to verify users' identities.
- Based on this architecture, this work aims to develop a pilot KYC implementation based on Portuguese Citizenship Card.
- Since Credibank [4] is a use case created by WalliD to show that the protocol is valid and valuable. The KYC produced must be implemented on Credibank system in order to identity their customers.
- This KYC solution can help WalliD as a company to build their own products using the protocol and this KYC solution.

Besides developing this platform, the research objective of this work intends to answer some of the following open research questions:

- RQ1: Can a blockchain, as a decentralized, immutable and transparent environment, be able to help to properly verify a citizen's identity?
- RQ2: Can the use of blockchain prevent the growing spread of copies of a citizen's identity documents over the internet?
- RQ3: Will the use of Digital Certificates improve the accuracy of the ID verification?
- RQ4: Can the use of a blockchain speed the KYC verification process?

These questions led to the development of the following research hypotheses:

- HYP1: The use of blockchain presents a secure and cyphered form of storing important information such as the personal identification, mainly because the only information being exchanged is in the form of anonymous hashes;
- HYP2: The use of blockchain not only allows a safe and private storage of data, but if properly configured, it also registers all accesses to that data, which allows

the user to have the information stored only once and to keep its access under control;

- HYP3: Much better than recognizing ink on a piece of paper, Digital Certificates ensure that the bearer of a specific identification has an identification recognized by a certified third-party (the best part of the real-world) and that it is also the person that submitted a particular request;
- HYP4: Reducing drastically the number of documents to ensure the veracity of the identity, automating the analysis of the identity and tying that analysis with a strong check of a Digital Certificate is a something that will potentially reduce the time required for identity verification to a fraction.

We consider that the confirmation of these hypotheses will satisfy the research questions being proposed, and the resulting system will provide its users with a safer, faster and more effective mechanism to validate identities both for the company and for the user being identified. The remainder of this document will then be spent finding evidences that support these hypotheses.

1.3 Publications and Exploitation

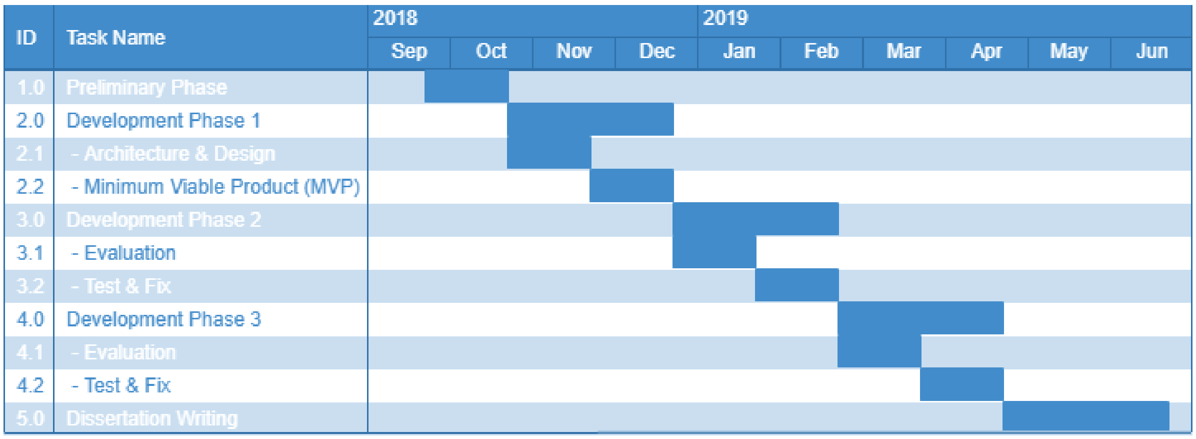
The work described on this dissertation resulted in a paper in the 9th IEEE-TEMS international Conference on Intelligent Systems, Madeira, Portugal, 2018, named as "WalliD: Secure your ID in an Ethereum Wallet" [5], and a paper in a special issue of the IEEE Instrumentation and Measurement Magazine[1] which has already been submitted and is waiting for approval.

Furthermore, the author of this dissertation is currently leading a team which is developing the first WalliD product. This product is being developed under a partnership with Docusign [6] and consists in a service that verifies a user's identity in order to prove that a signature in a document really belongs to him. In order to develop this solution, the proposed product uses the WalliD protocol to fetch users' identities and validate them through a KYC such as the one available at Credibank. In the end, the proposed solution will sign a document just after the user, in order to certify any entity that the user's signature really belongs to him.

[1] https://ieeexplore.ieee.org/xpl/aboutJournal.jsp?punumber=5289

1.4 Working Plan

The following Gantt chart in Figure 1 illustrates the plan which was followed during this work.



ID	Task Name	2018				2019					
		Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun
1.0	Preliminary Phase	■	■								
2.0	Development Phase 1		■	■	■						
2.1	- Architecture & Design		■	■							
2.2	- Minimum Viable Product (MVP)			■	■						
3.0	Development Phase 2				■	■	■				
3.1	- Evaluation				■	■					
3.2	- Test & Fix					■	■				
4.0	Development Phase 3						■	■	■		
4.1	- Evaluation						■	■			
4.2	- Test & Fix							■	■		
5.0	Dissertation Writing								■	■	■

Figure 1 - Working Plan

A more detailed description of each of the above tasks:

- **Task 1 (September and October):** Learning about project goals, related work and involved technologies. Preliminary report writing;
- **Task 2 (October, November and December):** Keeping the preliminary report updated, start designing the library architecture and its flow charts. Perform the implementation starting by the basics, such as JUnit testing and parsing the dataID JSON;
- **Task 3 (December, January and February):** Evaluating the current solution using a dummy dataID JSON. Performing some improvements and starting the integration tests;
- **Task 4 (February, March and April**): Performing some tests with real dataID JSON files. Testing the current implementation under pre-production scenario;
- **Task 5 (April, May and June):** Report writing.

1.5 Document Structure

The remainder of the document is structured as follows. Chapter 2 presents related work. Chapter 3 briefly describes how the WalliD protocol works. Chapter 4 talks about the design, architecture and the integration phase. Chapter 5 discusses achieved results. And, finally, Chapter 6 talks about future work and concludes.

Chapter 2

Related Work

This chapter reviews some systems, concepts and approaches that are somehow related to this work. Section 2.1 briefly introduces blockchains and how they are related to WalliD. Section 2.2 presents some other projects that aim to do the same as WalliD. Then, Section 2.3 introduces some approaches to automatize KYC services. Section 2.4 talks about X.509 certificates, and Section 2.5 presents how a Public Key Infrastructure for X.509 (PKIX) works. Then, Section 2.6 talks about a very important topic of this work which are the certification paths. Section 2.7 presents two ways which could be used to check whether a certain certificate is revoked or not. Then, Section 2.8 presents the Cryptographic Message Syntax (CMS), which is a standard to sign, digest, authenticate or encrypt arbitrary message contents. Finally, Section 2.9 talks about a standard which is used to store several cryptographic objects, such as private keys and certificates, in one only file.

2.1 Blockchain with WalliD

It is common knowledge that it is easier to copy a set of bits (digital money) than a piece of paper (money on the real world). One of the biggest issues related to digital money is the "double spending" problem: on the real world we are not able to spend more than once the same coin; however, since digital money are sets of bits, we are able to clone them and spend the same coin more than once. Satoshi Nakamoto explains this problem and explains how Bitcoin [7] deals with it. His solution uses a chronologically ordered timestamped ledger, also known as "blockchain", to solve this problem.

Bitcoin stores all money transactions on those blocks in order to allow everyone to check who is the owner of that money. Even if we copy a set of bits in order to duplicate money, everyone will be able to look at the blockchain and realize that we are not the owner of that piece of money – then the transaction will not be performed. This is possible because blockchains have a property called immutability, which prevents them from being changed without being noticed. This is possible because each block has a special pointer which references the previous block. This pointer is calculated through a hash function. Basically, hashes are functions with certain special properties which calculate digests through a given input. On this case, the input of the hash function is the entire block. Here is where the "mining" process comes in. Each block is only allowed to join the blockchain if its digest starts with a certain number of zeroes, being that number of zeroes defined by the network. In order to find the digest which fulfils this condition, each block has a number which is computed in a brute force way. So, the mining process consists of (*i)* adding a certain number (called "nonce") to the block being added, *(ii)* computing its digest, and *(iii)* checking if that output contains the expected number of zeroes. If it doesn't, this process must be repeated once again. This randomized concurrency control mechanism process is called "Proof of Work" (PoW) and aims to establish consensus in the network. The difficulty level is adjusted dynamically and grows exponentially with the number of zeroes at the left required by the network. For security reasons, Bitcoin's network is configured to append a new block in an average of 10 minutes. PoW is a technique which works very well on networks with a large number of nodes, but classic protocols such as Practical Byzantine Fault Tolerance (PBFT) [8] are unfeasible because they exchange a lot of messages. On the other hand, PoW requires a huge quantity of energy consumption, which is a disadvantage of this technique.

Unlike Bitcoin, that can only be used to exchange money, the Ethereum [9] network allows running code over its blockchain through smart contracts. While a standard contract outlines the terms of a relationship (usually one enforceable by law), a smart contract enforces a relationship with cryptographic code as described on Ethereum white paper [10]. Those smart contracts, or "autonomous agents", as they are called, can remove third parties and bring a layer of trust over the network layer. For instance, usually when we buy something on the Internet, we would like to ensure that no one will misbehave. In order to do that, people trust some companies (for example, Paypal) as third parties to deal with this problem. If any problem occurs, we can contact them and ask our money back. We can code a set of rules in a smart contract which will perform the same behavior as PayPal.

The process of mining in the Ethereum network is similar to the one described for Bitcoin in Satoshi Nakamoto's white paper – where a "miner" tries to find a particular nonce which is added to the block. Then, the miner must generate a hash of that block and check if it contains a certain number of zeroes at the left. Only the blocks which accomplish this condition can be added to the blockchain.

The Bitcoin protocol was made specifically for money exchanging, so it does not fit WalliD's needs; WalliD's protocol was built on top of the Ethereum blockchain. As this dependence on Ethereum is a disadvantage, there are some solutions such as Syahputra and Weigand that have developed a process that can generate smart contracts for heterogeneous blockchain technologies, such as Ethereum or Hyperledger, using Unified Modelling Language (UML) and Object Constraint Language (OCL) to implement the workflow and algorithm used to produce smart contracts in a platform-independent way, and proposed algorithms to simplify the development for heterogeneous blockchain platforms [11]. This could be very interesting if, in the future, the project faces limitations regarding the Ethereum protocol, and this approach could be adopted in order to use other blockchains.

2.2 Similar Projects to WalliD

Some other projects similar to WalliD were also developed during the last couple of years, their main focus being the use of the blockchain as a tool to let users keep control of their identities and transactions.

One of them is named CIVIC project [12], and its main difference to WalliD is that CIVIC does not actually connect to any document or certified system – it simply asks that information to the user and assumes the inserted information to be correct. This means that it may be useful for dealing with KYC processes, but misses one essential feature which is to avoid fake impersonation of another person's identity.

On the other hand, the uPort project [13] also deals with the decentralization of identity management, but their approach is to create a separate, independent identity system with their own security and not bound to real documents. Again, the main problem is about the trust and actual effectiveness of having an identity system that is not recognized by everybody.

The Persona project [14] is a solution for identity management which is also aligned with the latest data protection regulations. Its scope is to empower the individual and grant them the control over their personal data as well as the means to secure access to their private details. The idea is again to allow users to insert identification attributes in the blockchain environment, and then to validate them by the user itself, so once again it is the users who are in charge of inserting the data that will state their identity and they will themselves validate if that data is correct. However, this will not prevent a user from faking, stating to be another person, and that is one major drawback of these systems.

Other similar projects do the same such as IDCoins [15], SelfKey [16], TheKey [17], and REMME [18], implementing authentication systems based on blockchain-based data. Existing electronic ID systems like Eestki and Guardtime from Estonia [19] are becoming leading environments for building European enterprise security solutions, as well as Bitnation [20]. Despite the success of some of these projects, the issue of connecting the already trusted offline identities with the blockchain and ultimately with online services remains.

2.3 KYC Services

There are some companies which use machine learning to provide KYC services. Jumio [21] developed a product named Netverify which uses Artificial Intelligence (AI) to prove its users' identities. In order to do that, the user can take a photo of a document asserting his identity, such as a driver license or passport. On the next step, the AI algorithm will analyze the document and will perform a biometric facial recognition through the user's smartphone camera. Shufti Pro [22] also provides KYC services through AI such as Jumio. They have a very interesting product which allows us to verify a user's handwritten notes. This could be used for instance to replace notaries.

On another approach, there are solutions based on Multi-factor authentication [23]. On that paper, the authors describe a KYC mechanism which, on a first stage performs a simple authentication with user's identity and a password. Afterwards, the KYC process executes an algorithm of risk analysis which results in a certain level of risk. Final stage of the verification is OTP / EMAIL / OTP & Email confirmation if it is indicated by the result of the risk level.

Finally, but not less important, there are similar solutions which are based on Big Data [24]. Researchers in India realized it is quite hard for bank institutions to identify unique customers. A person is able to open a bank account using several different documents, like for instance Permanent Account Number (PAN), passport and even with driving license. Since all these documents have different serial numbers, it is very hard to find a unique identity; they also show an example of a customer with five different records. A way to mitigate this issue is to use the first name, last name, birth date, address and identification to match a customer. The proposed solution on this paper uses a fuzzy matching technique, an advanced mathematical process that determines the similarities between data sets, information, and facts – where the outcome is neither true or false, nor 100 percent certain. The process compares any data type of any length and from any place in a field to find non-exact matches. For every piece of data examined, the fuzzy matching process will give a probability score to determine the accuracy of the match.

2.4 X.509 Certificates

X.509 certificates give confidence to the users of a public key that the private key is owned by the correct subject. This confidence is achieved through the use of public key certificates which are data structures that bind public keys to subjects. This binding is performed by a trusted CA which digitally signs each certificate. Each certificate contains the following fields:

- **Version:** Indicates the certificate version;
- **Serial Number:** Holds a unique number per certificate;
- **Signature:** Contains the issuing authority signature;
- **Issuer:** Indicates the issuer's distinguished name;
- **Validity:** Contains the activation and expiration dates;
- **Subject:** Indicates the subject's distinguished name;
- **Extensions:** Are fields which are only available in version 3 certificates. This property allows applications to add some arbitrary for specific purposes which are not covered here.

There are two main methods to encode these certificates which are:

- **Distinguished Encoding Rules (DER):** Binary encoding for certificate data;
- **Privacy-enhanced Electronic Mail (PEM):** Base64 encoding of DER encoded format, with header and footer lines added.

The use of digital certificates is strongly recommended; it provides us with some guarantees that we are dealing with the right subject. The X.509 certificates will be intensively used in this work since we are dealing with electronic identity documents, such as the Portuguese citizenship card.

2.5 Public Key Infrastructure for X.509 (PKIX)

The Public Key Infrastructure (PKI) for X.509 certificates [25] profiles the format, the semantic of certificates and Certificate Revocation Lists (CRLs) for the Internet PKI. The goal of this specification is to facilitate the use of X.509 certificates within Internet applications such as e-mail, user authentication, and so on.

The architecture model of PKIX assumes the existence of five entities:

- **End Entity (EE):** The user or system which owns a PKI certificate that is the subject of a certificate;
- **Certification Authority (CA):** Trusted entity which issues and revokes certificates;
- **Registration Authority (RA):** Optional entity to which a CA delegates certain management functions, such as user management and logging of certain events;
- **CRL issuer:** System that generates and signs the CRLs;
- **Repository:** System or collection of distributed systems where the certificates and CRLs are being stored in order to distribute them to the end entities.

Figure 2 illustrates how these entities communicate with each other.

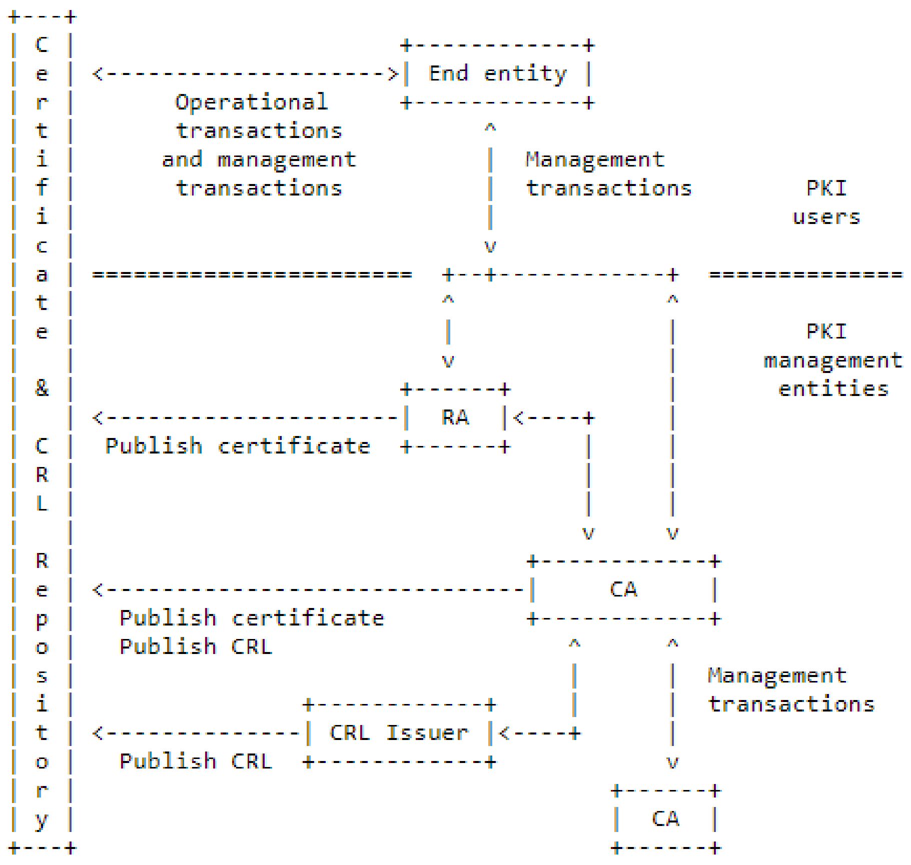


Figure 2 - PKI Entities [25]

Since Portuguese citizenship cards use X.509 certificates, we will consider this architecture during this work. In this case, Portuguese citizens behave as PKI users, and the Portuguese mint house, which is the entity that issues the certificates within the Portuguese citizenship card, acts as PKI management entity.

2.6 Certification Paths

In order to validate a certain certificate, we need to perform some verifications, such as the validity and the issuer's signature, among others. But sometimes there are certificates that are not issued and signed by a CA, but by an intermediate entity. On these cases, we may need more than one certificate to perform the certificate validation. Commonly, the aggregations of these certificates are called certificate chains, or certification paths.

A certificate must not be present more than once in a certain certification path. However, if the trust anchor is passed to the certification path as a self-signed certificate, that certificate does not make part of the prospective certification path. A trust anchor is a special kind of certificate where the trust is assumed and not derived.

These certification paths will be used on this research to validate the user's certificates which are within the Portuguese citizenship cards. Without this verification we cannot trust a certain certificate; through this method we are able, for instance, to identify/detect forged certificates. On those cases, the certificates will not be issued by a trusted anchor, so in the end we can say that a certain end entity certificate is also not trustful.

This procedure is described in a more detailed way on section 6 of RFC5280 [25].

2.7 Revocation Lists

CAs are responsible for indicating the revocation status of a certain certificate. Usually, this information is provided using Certificate Revocation Lists (CRLs) [25] or Online Certificate Status Protocol (OCSP) [26].

CRLs are lists of revoked certificates that are issued by CRL issuers, which are also CAs or entities that were authorized by a CA to issue CRLs. Each element of those lists contains the serial number of the revoked certificate and the date of revocation. Figure 3 illustrates an example of a CRL content.

```
Revoked Certificates:
    Serial Number: 2572757EAAF2BEC5980067579A0A7705
        Revocation Number: May 1 19:56:10 2019 GMT
     Serial Number: 776DDD15D25C713616E7D4A8EACFB41A
        Revocation Number: May 10 13:03:16 2019 GMT
```

Figure 3 - CRL content

A disadvantage of CRLs is that, in order to check whether a certain certificate is revoked, it is necessary to download the entire CRL, which contains several certificates that will be useless. Of course, this introduces a huge overhead. Another disadvantage regards scalability – since a CRL contains several revoked certificates, applications will perform a search in order to check whether a certain certificate is revoked or not. And, maybe the worse one, CRLs are usually not updated every day. This means that there is no guarantee that the certificate of interest is revoked or not.

In order to solve some of these constraints, a new protocol named OCSP replaced CRLs. First of all, when an application asks an OCSP server to check whether a certain certificate is revoked, the server checks and gives the status for that singular certificate. The protocol defines three kinds of status:

- **Good:** This status indicates a positive response about the requested certificate, meaning that the certificate is not revoked.
- **Revoked:** The revoked status means that the requested certificate is revoked.
- **Unknown:** This status indicates that the server does not know if the certificate is revoked or not.

Figure 4 illustrates a sample of an OCSP response.

```
Response verify OK
0x25F5V12D5E6FD0BD4EAF2A2C966F3B4aE: good
    This Update: May 1 19:56:10 2019 GMT
    Next Update: May 10 13:03:16 2019 GMT
```

Figure 4 - OCSP content example

Since we are dealing with highly volatile certificates, such as the Portuguese citizenship card where a certificate could easily get revoked, these two mechanisms will be present in this work – in case OCSP is not available, this solution is able to use a cached CRL if available.

2.8 Cryptographic Message Syntax (CMS)

Cryptographic Message Syntax (CMS) [27] is an IETF standard syntax based on PKCS#7 [28] used to digitally sign, digest, authenticate, or encrypt arbitrary message content. CMS is general enough to support many different content types. There are six content types that describe the form enhancement that could be applied to the digital data:

- **Data:** this content type is intended to refer to arbitrary octet strings, such as ASCII text files. Typically, this type is used on the other types;
- **Signed-Data:** a content of any type together with encrypted message hashes of the content for zero or more signers;
- **Enveloped-Data:** a content of any type together with encrypted content encryption keys for one or more recipients;
- **Digested-Data:** a content of any type together with the message digest and the digest algorithm used;
- **Encrypted-Data:** this type contains an encrypted content of any type, however, unlike the enveloped-data, it does not contain the recipients, nor encrypted content keys – those keys must be managed by other means;
- **Authenticated-Data:** a content of any type together with a Message Authentication Code (MAC), and encrypted authentication keys for one or more recipients.

2.9 PKCS#12

PKCS#12 [29] is a standard which describes a transfer syntax for personal identity information such as private keys, certificates, etc. Applications that support this standard allow the user to import, export and exercise a single set of personal identity information.

Furthermore, PKCS#12 describes a very useful standard which allows us to store in only one file a set of private keys, certificates; usually we call this a *keystore*.

This feature is very interesting for this work since we are aiming at building a generic solution. In this case, our KYC system will receive a keystore which contains all certificates (intermediate and root) needed to build certification paths. These paths will be used to verify the X.509 certificate which is associated to the Portuguese citizenship card. For future nationality integrations or certificate updates this could be very useful; we can build a keystore with all that is needed to validate the chain in only one file, without care about their codification types.

Chapter 3

The WalliD Protocol

WalliD is a new Ethereum-based protocol that allows citizens to encrypt and store their certified identity documents and attributes from the real world on the blockchain, access and instantly exchange them with KYC services they trust. WalliD is described on a white paper [30] and it is supported by open source software whose code is available on GitHub [31]. The main innovation in this protocol is the ability to verify if the uploaded attributes are under a globally trusted digital certificate such as the X.509, making sure that every identity uploaded and verified by the system is a valid identity and certified by its Certification Authority (CA).

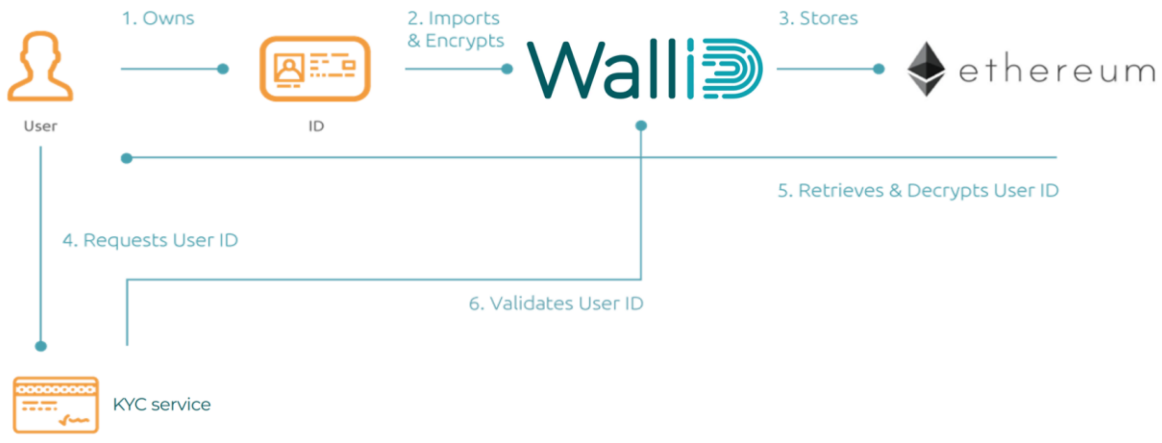


Figure 5 - WalliD Flow [30]

The main steps of the WalliD flow are:

- **Step 1:** First of all, the user must own an identity document which must contain an X.509 certificate within;
- **Step 2:** The user exports his identity attributes from his document. This process requires a computer and a smart card reader;
- **Step 3:** The user stores his identity attributes previously exported in the WalliD infrastructure.

The onboarding phase, which is performed by all the three steps above, is only executed once by the user. However, if the user asks for a new identity card, this process must be repeated. This happens because the X.509 certificate is not valid anymore. Then, the next three following steps are executed once the user wants to prove his identity:

- **Step 4:** The system that needs to validate the user's identity asks for his identity;
- **Step 5:** The user asks for his identity to the WalliD smart contract, receives it and decrypts it. Then, the user's identity is sent to the system that asked for it;
- **Step 6:** The system receives the user's identity and runs his KYC service to validate it.

The following sections describe some components that make part of WalliD architecture. Section 3.1 talks about MetaMask which is a web plugin used to interact with the Ethereum blockchain. Section 3.2 describes the ImportiD component which is used to extract and generate the WalliD data structure from the Portuguese Citizenship Card. Section 3.3 presents MyEtheriD which is responsible to store the data structure generated by ImportiD in the WalliD infrastructure. Section 3.4 describes another WalliD component named StoreiD, which is responsible to store part of the data generated by ImportiD, such as the user's X.509 certificate extracted from his Portuguese Citizenship Card. Finally, but not less important, Section 3.5 describes a use case created by the WalliD team in order to demonstrate a complete round trip around the protocol.

3.1 MetaMask

MetaMask [32] is a cryptocurrency wallet which can be used on Chrome, Firefox and Brave browsers. Since the web browser does not know how to deal with the Ethereum blockchain, MetaMask is installed as a plugin which injects a JavaScript library named Web3 in order to provide the communication between the web browser and the Ethereum network. It allows users to sign smart contracts, and interface with Ethereum distributed applications without running a full node.

All WalliD users must have a MetaMask account; it is through their wallet address that the association is established to his digital identity.

3.2 ImportiD

The ImportiD [33] deals with the data identity extraction from the user's document, such as the Portuguese citizenship card. In order to achieve this goal, the user must own a smart card reader that will be used to read the user's identity document.

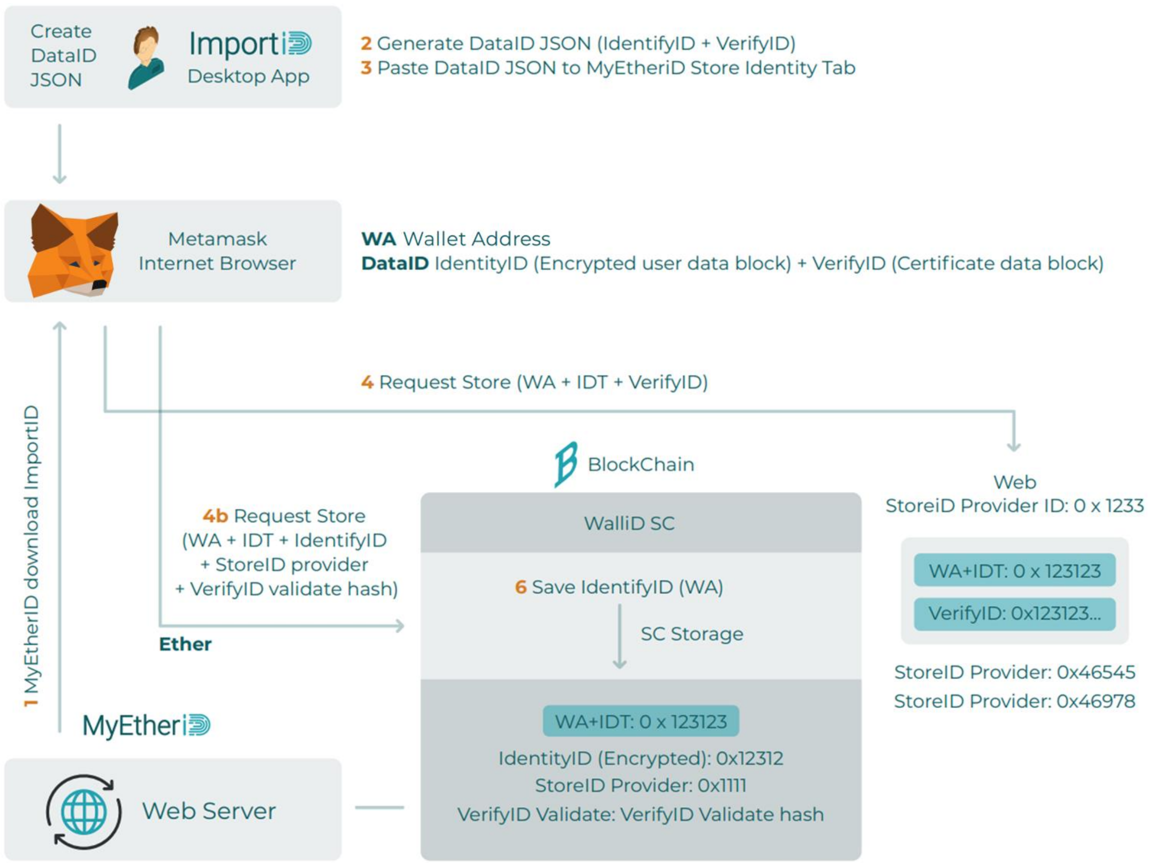
Since each identity has its own data format, it is not possible to develop a generic ImportiD that could deal with all identity documents. Each document will have its specific ImportiD in order to perform the data extraction. As a result, ImportiD will generate a JSON file which contains all information regarding the user's identity. This JSON is named dataID and contains three main blocks of data. The first one is called identifyiD and contains the user's identity attributes, such as the first name, last name, address, among others. The second one is called verifyID – this block contains the X.509 certificate and a Document Security Object (SOD) [34] file which is also present in Portuguese citizenship cards. Without this block, the user's identity attributes have no meaningful value. The SOD is used to verify the user's identity and address attributes, while the X.509 certificate is used, for instance, to check if this identity is still valid. If a citizen loses its citizenship card, the action of requesting another one causes this certificate to be automatically revoked by the governmental institution which manages the Portuguese identities.

In order to perform a match between these two blocks, the ImportiD component signs the user's wallet address and stores it in the verifyID block. This signature can be verified through the digital certificate which is stored on the verifyID. Finally, but not less important, the third element on the dataID JSON file is the StoreIDProvider which defines the provider that will store verifyID block.

3.3 MyEtheriD

While the user's attributes are being stored in the Ethereum blockchain encrypted with its MetaMask public key, the verifyID data is stored in another component named StoreiD [35].

The component which is responsible for storing this information is named MyEtheriD [36] [37] and receives as input the JSON file generated by ImportiD. The process how WalliD fetches and stores user's identities is illustrated in Figure 6.



Create DataID JSON
ImportiD Desktop App
2 Generate DataID JSON (IdentifyID + VerifyID)
3 Paste DataID JSON to MyEtheriD Store Identity Tab
Metamask Internet Browser
WA Wallet Address
DataID IdentityID (Encrypted user data block) + VerifyID (Certificate data block)
4 Request Store (WA + IDT + VerifyID)
1 MyEtheriD download ImportID
BlockChain
Web StoreID Provider ID: 0 x 1233
WA+IDT: 0 x 123123
VerifyID: 0x123123...
StoreID Provider: 0x46545
StoreID Provider: 0x46978
4b Request Store (WA + IDT + IdentifyID + StoreID provider + VerifyID validate hash)
Ether
WalliD SC
6 Save IdentifyID (WA)
SC Storage
WA+IDT: 0 x 123123
IdentityID (Encrypted): 0x12312
StoreID Provider: 0x1111
VerifyID Validate: VerifyID Validate hash
MyEtheriD
Web Server

Figure 6 - Extracting and Storing identity flow [30]

The main steps of the MyEtheriD flow are:

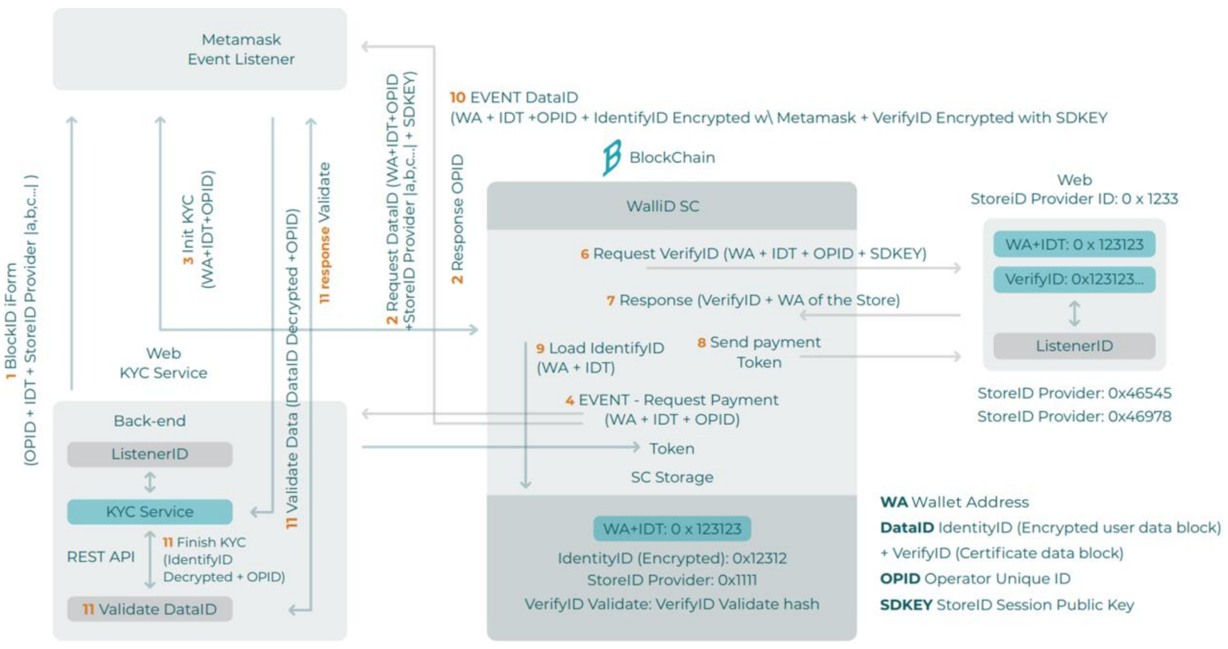
- **Step 1:** The user opens a web browser and navigates to MyEtheriD.io PoI provider to download an importing application such as ImportiD;
- **Step 2:** Then, the user opens his importing application, follows the instructions and generates his dataID JSON;
- **Step 3:** The user goes back to his web browser, accesses MyEtheriD.io again and pastes the generated dataID block into the “Store Identity” tab;
- **Step 4 & 4b:** User presses “Connect with MetaMask” to store his dataID. Then, the verifyID block will be stored in the StoreiD provider (4). Hence, the identityID block will be stored in the blockchain through WalliD smart contract (4b);
- **Step 5:** The stored identityID block is stored in the smart contract and indexed through the user’s wallet and Identity Type (IDT).

3.4 StoreiD

In order to turn WalliD profitable, the verifyID block is provided by a third-party, generally out of the chain, such as a regular database system. The identity attributes have no meaning without the certificate, wallet address signature and the SOD – it is through these data that we can verify the trustiness of user’s identity. In order to get the verifyID block, the entity that wants to verify the user’s identity, such as Credibank, must pay a fee to the entity which is storing the verifyID block. This fee is charged in WALs (currency used in WalliD) per each time that an entity such as Credibank wants to perform a request to receive a verifyiD block. This payment is intermediated by WalliD’s smart contract, and if the entity does not receive the verifyID block within a certain period of time, the payment is refunded.

3.5 Use Case

A simple use case was built to test the WalliD protocol. This use case includes a dummy bank institution called Credibank. The goal of this institution is to lend money to their customers. In order to speed up their internal identification processes, Credibank uses WalliD to verify the identity of their customers. The flowchart of this use case is illustrated on Figure 7.



Metamask Event Listener
1 BlockID iForm (OPID + IDT + StoreID Provider [a,b,c...])
3 Init KYC (WA+IDT+OPID)
11 Validate Data (DataID Decrypted +OPID)
11 response Validate
2 Request DataID (WA+IDT+OPID +StoreID Provider [a,b,c...] + SDKEY)
2 Response OPID
10 EVENT DataID (WA + IDT +OPID + IdentifyID Encrypted w\ Metamask + VerifyID Encrypted with SDKEY
BlockChain
WalliD SC
6 Request VerifyID (WA + IDT + OPID + SDKEY)
7 Response (VerifyID + WA of the Store)
9 Load IdentifyID (WA + IDT)
8 Send payment Token
4 EVENT - Request Payment (WA + IDT + OPID)
Token
SC Storage
WA+IDT: 0 x 123123
IdentityID (Encrypted): 0x12312
StoreID Provider: 0x1111
VerifyID Validate: VerifyID Validate hash
Web StoreiD Provider ID: 0 x 1233
WA+IDT: 0 x 123123
VerifyID: 0x123123...
ListenerID
StoreID Provider: 0x46545
StoreID Provider: 0x46978
Web KYC Service
Back-end
ListenerID
KYC Service
REST API
11 Finish KYC (IdentifyID Decrypted + OPID)
11 Validate DataID
WA Wallet Address
DataID IdentityID (Encrypted user data block) + VerifyID (Certificate data block)
OPID Operator Unique ID
SDKEY StoreID Session Public Key

Figure 7 - Proof of Identity flow [30]

The main steps to proof user's identity are:

- **Step 1:** The customer opens a web browser, logs in on his MetaMask account and then navigates to the Credibank web page. Then, the customer must fill the credit purpose, the amount of money and the payment tranches. Each identification process (KYC) has an operation unique identifier (OPID), and a list of StoreiD providers which has a list of identity documents (IDT) associated. The customer must choose which provider and document will be used on the identification process.
- **Step 2:** Once selected the provider and the document, the WalliD smart contract will be invoked asking the user's dataID block. At that moment, MetaMask will show a popup asking the user to validate the operation. Once accepted, the call which invokes the smart contract contains a public session key (SDKEY). That public session key is generated by the client in order to be used to encrypt the verifyID block which is stored on StoreiD. This key will be used by StoreiD in order to encrypt the verifyID block. Since the private key is stored on the user's side, only him will be able to decrypt it. This is used to protect the communication between the user and the StoreiD.
- **Step 3:** Credibank back-end is listening for all events that require for a KYC verification. In order to filter which events belong to it, the service that deals with that task needs the user's wallet address (WA), the IDT and OPID.
- **Step 4:** The smart contract then triggers an event in order to request a payment.
- **Step 5:** Credibank back-end receives the event and performs the payment which is performed in WALs.
- **Step 6:** When the payment is received by the smart contract, it checks which StoreiD provider customer was chosen and asks for its corresponding verifyID block. In order to find the correct verifyID block, the StoreiD needs the WA and the IDT. This dependency is related with the way how StoreiD stores data, where each verifyID entry is indexed by these two parameters.
- **Step 7:** **S**mart contract collects the verifyID block plus the StoreiD wallet address from StoreiD. These data were encrypted using SDKEY provided previously.

- **Step 8:** Once the data is collected, the WalliD smart contract pays to the StroreiD.
- **Step 9:** The smart contract then loads from the blockchain the identityID block which is encrypted by MetaMask, ensuring that only the customer will be able to decrypt it. This encryption process was performed during the onboarding process.
- **Step 10:** WalliD smart contract emits the DataID event which contains the identityID and the verifyID blocks.
- **Step 11:** The service provider web page will receive this event and ask the user to decrypt that data using MetaMask. Once successfully decrypted, the web page will send that information to a component that will validate the user's identity.

3.6 Summary

As presented on the previous sections, WalliD aims to solve a lack of trust on the online world bringing identities digital certified used on the physical world, such as the Portuguese Citizenship Card. As illustrated on Figure 8, in order to accomplish this goal, WalliD stores the users' identities on their Ethereum wallet.

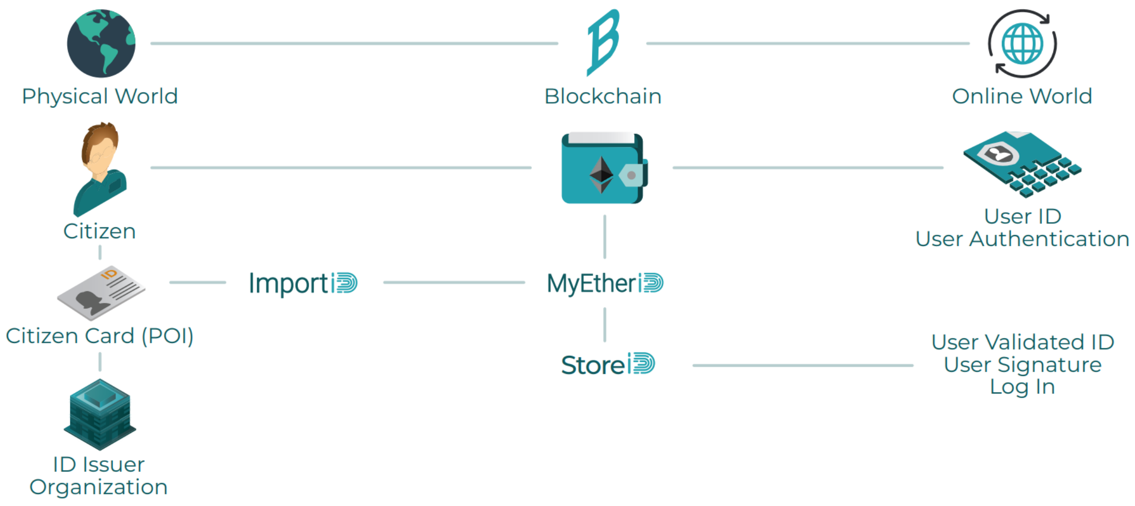


Figure 8 - WalliD ecosystem

Since that document was issued by a trustful entity and contains a digital certificate within, it should be possible to verify that identity on the online world through it. The main goal of this work is to develop an architecture, and its implementation, for the component referred to on step 11, which is responsible to verify this identity on the online world.

Chapter 4

Design and Implementation

During this work, several cryptographic techniques were used, such as signature verification, certificate chain verification, among others. Since these techniques are widely used, a research task was performed in order to find some open source libraries that could be used. During that task, the author realized that several developers were using a library called Bouncy Castle [38]. This library completely fulfills this project needs: it is open source, it offers the features needed to develop this work and it has a nice community which can provide pretty good support.

Then, after reading the documentation, the author realized that this library is available for Java which is the language he is most familiar with. The output of this implementation will result in a library in a Java Archive (JAR) format because it gives the developers more flexibility than a web service. For example, if the output is a web service, a developer will not be able to develop a socket-based solution.

This software was designed following a modular architecture in such a way that it will be easy to add more KYCs, based on other kind of identities like the Spanish citizenship card, the Portuguese mobile digital key [39] and so on. In order to do that, this software was developed using an open source framework developed by Google called Guice [40]. Guice helps developers to use Dependency Injection (DI) patterns in order to make their code very modular.

Test-Driven Development (TDD) methodology was also used in this work in such a way that the development is supported and validated by integration and unitary automated tests. In order to do that, some Java libraries are used such as JUnit [41]. These tests help a lot, especially during the deployment phase because it is possible to test any feature automatically in any moment, as well as to ensure business continuity and continuous integration.

4.1 Architecture

The designed architecture is represented on Figure 9. This and the following sections will detail each of its components and flows.

First of all, the flow starts when the system receives the dataID block, which arrives at a gateway. The dataID is a cleartext block of data, which was already decrypted by the user through his MetaMask account. Then, the Portuguese CC Gateway is responsible for dealing with user requests and responses; furthermore, it dispatches the dataID block to three components named verifiers, which, as the name says, are components that perform some verifications on the received data. The main ones are:

- **Certificate Verifier:** This module builds the certificate chain and verifies if the certificate is revoked or expired.
- **Wallet Address Verifier:** The verifyID block contains a signature of the user's wallet address. This module verifies this signature in order to check if this wallet really belongs to the user.
- **SOD Verifier:** This module verifies the authenticity of the user's identity and address attributes.

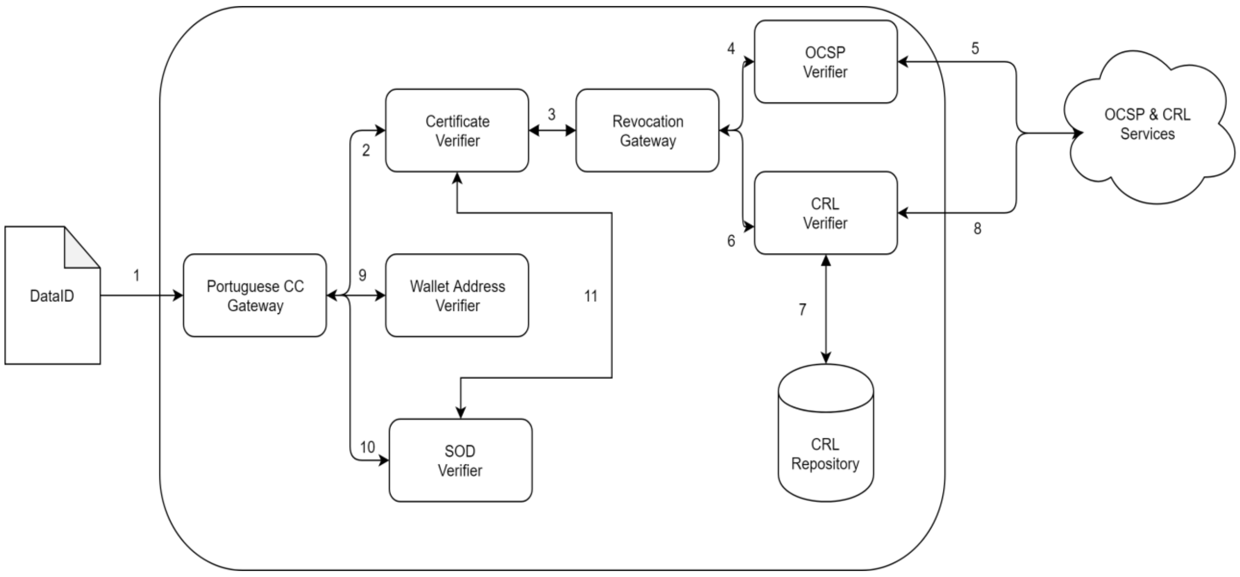


Figure 9 - Proposed KYC Architecture and Flows

At the beginning, the flow is dispatched from the gateway to the certificate verifier component. At this phase, there are some validations performed, such as the certificate validity and the certificate path chain. If everything goes without failures, the user's certificate is dispatched to the revocation gateway. This gateway is responsible to dispatch the revocation verifier request. Since OCSP is getting updated more frequently than CRLs, and the overhead is lower as well, the requests are dispatched first to the OCSP verifier. If the OCSP request fails, or the response is invalid, the revocation gateway dispatches the request to the CRL verifier. Before asking a new CRL remotely, the verifier checks if that CRL is available locally cached and his validity. This procedure avoids nonsense network communications since the local CRL is still valid.

The next verification to be done is the wallet address. First of all, this module verifies if the wallet used has the correct address stored in the blockchain. Then, the next step is to verify the wallet address signature which is executed by the ImportiD component during the onboarding process.

Finally, the flow goes to the SOD verifier. The main goal of this component is to validate the user's identity and address attributes. This verification is performed through a pair of hashes SHA256 which are stored in the SOD file. If this verification succeeds, a secondary verification is performed, which is related with the X.509 certificate that is in the SOD file. This certificate verification follows the same flow as the user's certificate which was presented previously.

If all these steps are performed with success, the user receives a positive feedback from the KYC module.

The following sections will explain in detail how each of these modules work.

4.1.1 Portuguese CC Gateway

The gateway module is the core of this solution. It is responsible for receiving users' requests, dispatch them to its sub-modules, and return the reply to the user. The flow chart illustrated on Figure 10 presents how this component processes users requests.

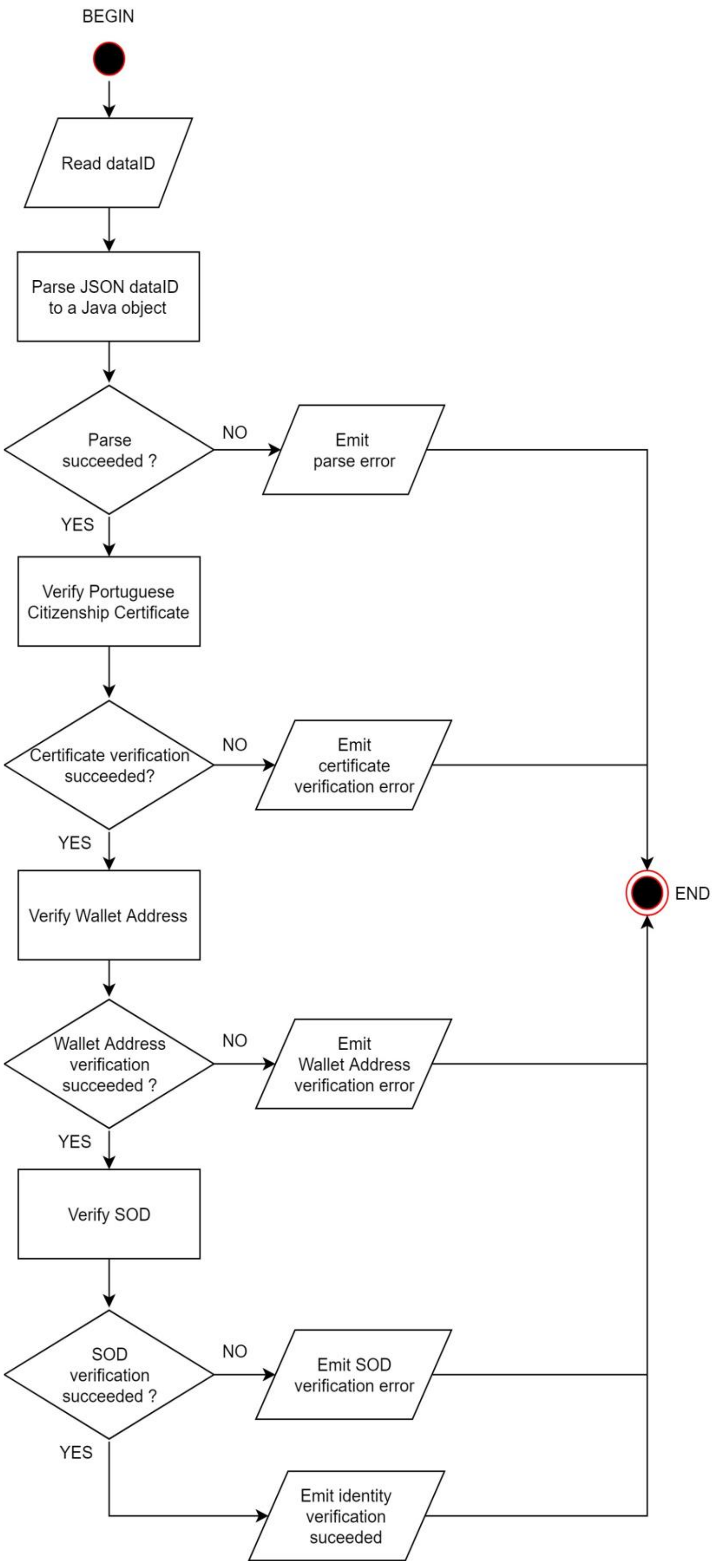

BEGIN
Read dataID
Parse JSON dataID to a Java object
Parse succeeded ?
NO
Emit parse error
YES
Verify Portuguese Citizenship Certificate
Certificate verification succeeded?
NO
Emit certificate verification error
YES
Verify Wallet Address
Wallet Address verification succeeded ?
NO
Emit Wallet Address verification error
YES
Verify SOD
SOD verification succeeded ?
NO
Emit SOD verification error
YES
Emit identity verification suceeded
END

Figure 10 - Main Flow

First of all, the main module receives a dataID block which is in the JSON format. Then, this block of data is parsed to generate a set of objects in order to make the data manipulation easier. If these objects are created successfully, the next step is to verify the certificate which is provided with the Portuguese citizenship card that is also stored in the verifyID block.

If this verification succeeds, the next verification to be done is the wallet address. This verification is mandatory because we need to ensure that the wallet really belongs to the user. Since this verification is performed through the user's certificate, this process can only be executed after the certificate verification. The reason for this is related with the way the wallet address is verified. During the onboarding process, the ImportID signs the user's wallet address using the private key stored in the Portuguese citizenship card. Then, the signature and the wallet address are stored in the dataID block, and the verification of the wallet address is executed by verifying this signature. Since this signature is verified by the public key which is stored inside the Portuguese citizenship card certificate, we must be sure that the certificate is already verified.

Finally, the last verification to be done is the SOD. This verification is optional except for cases where all identity and address attributes are verified. If only a small set of attributes such as name, birth date, the certificate verification already ensures this verification. If that is the case, the flow can be stopped, and some resources and time saved; otherwise, this module can be used to verify the remaining attributes.

4.1.2 Certificate Verifier

The certificate verifier is responsible for the X.509 certificate verification. The role of this module is to verify whether a certain certificate is trustful or not. The flowchart illustrated in Figure 11 shows how this verification is performed.

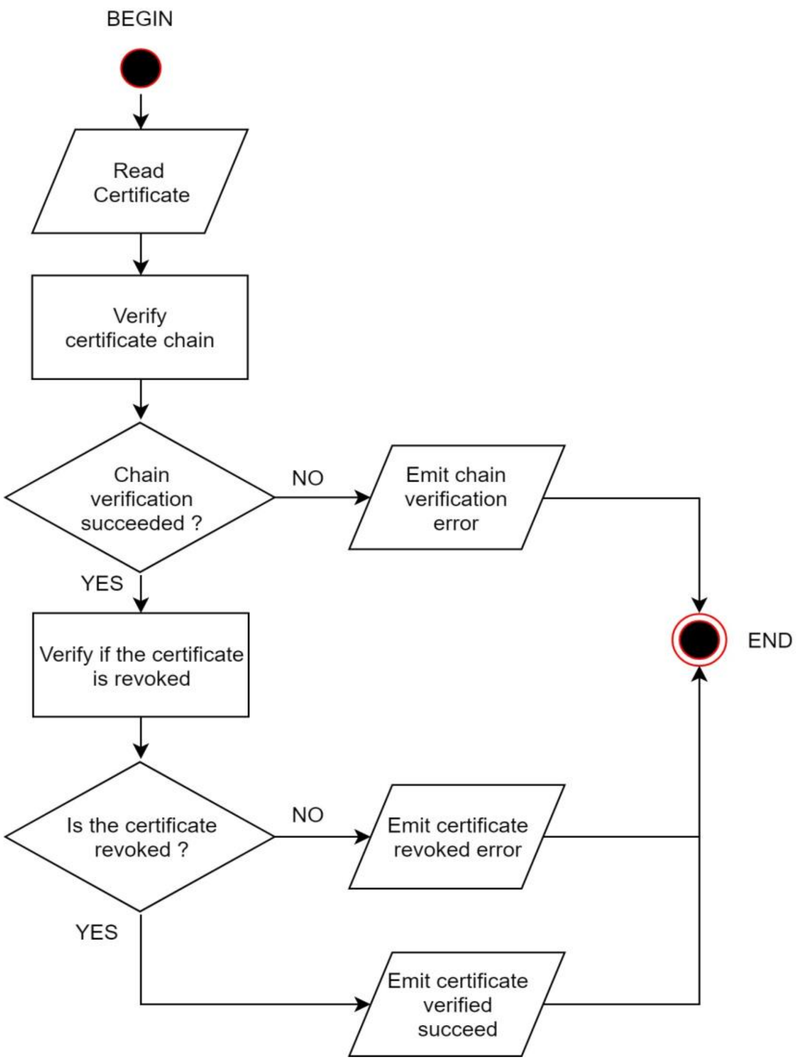

BEGIN
Read Certificate
Verify certificate chain
Chain verification succeeded ?
NO
Emit chain verification error
YES
Verify if the certificate is revoked
END
Is the certificate revoked ?
NO
Emit certificate revoked error
YES
Emit certificate verified succeed

Figure 11 - Certificate Verification Flow

In order to achieve its goal, the certificate verifier uses two modules which are responsible for the following tasks:

- **Chain verification:** This process aims to build a chain of trust in order to check if the certificate is trustful or not;
- **Revocation verification:** There are some cases for which this verification must fail even if the chain is built with success. For instance, if a citizen loses his citizenship card, he will ask for another one. The governmental institution which deals with this process will revoke the certificate which is in the old, lost card. This process deals with this situation, basically verifying if the citizen certificate is revoked or not.

4.1.3 Certificate Chain Check

The certificate chain check module is responsible for building a certification path between the end entity and the certification authority. This procedure is explained on Figure 12.

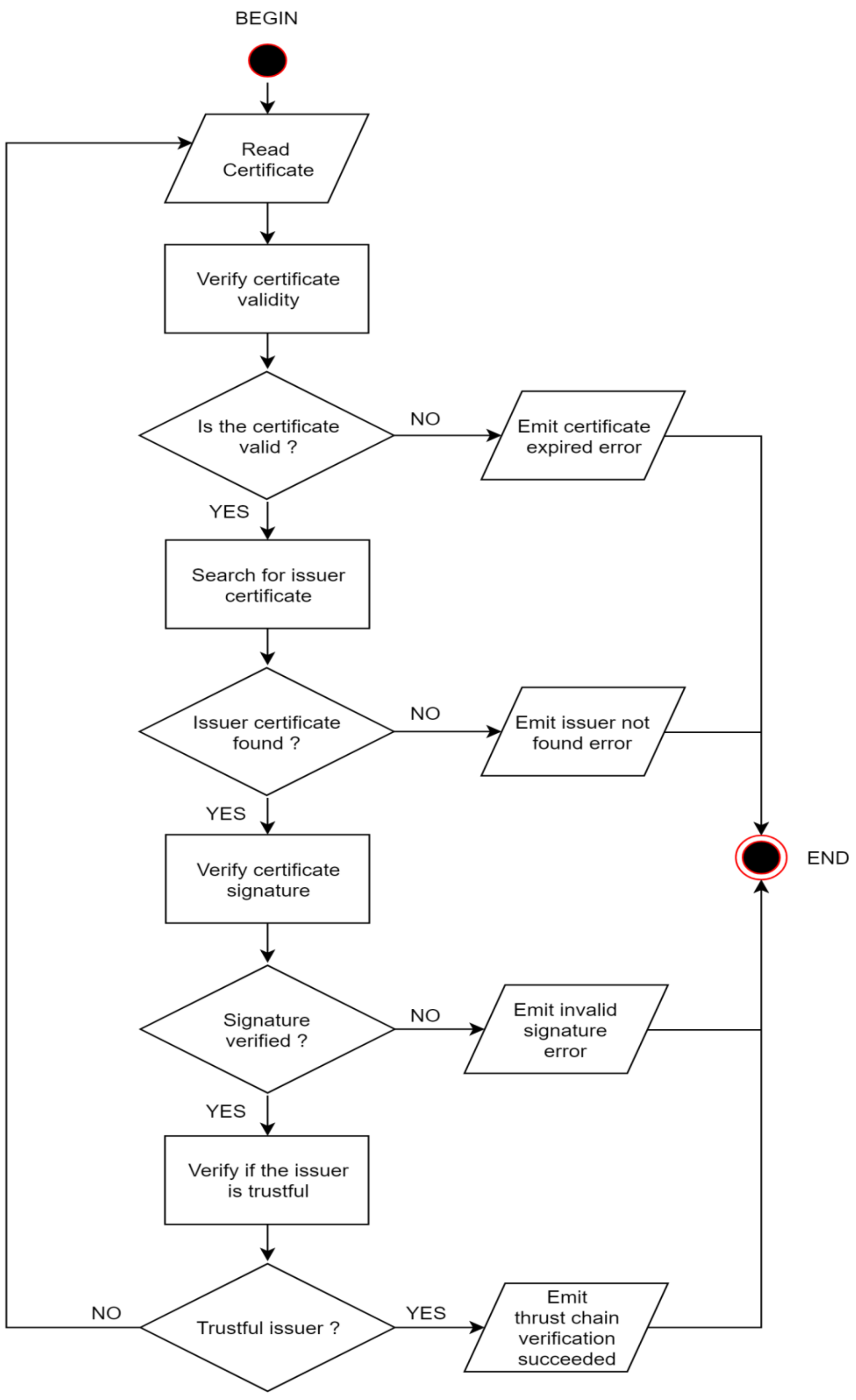

BEGIN
Read Certificate
Verify certificate validity
Is the certificate valid ?
NO
Emit certificate expired error
YES
Search for issuer certificate
Issuer certificate found ?
NO
Emit issuer not found error
YES
Verify certificate signature
END
Signature verified ?
NO
Emit invalid signature error
YES
Verify if the issuer is trustful
NO
Trustful issuer ?
YES
Emit thrust chain verification succeeded

Figure 12 - Chain Verification Flow

The next items will explain how each of these verifications are performed:

- **Validity verification:** First of all, the process needs to ensure the certificate is not expired. For instance, a certificate which has an expiration date of yesterday must miss this verification at the next day. In order to do that, the process compares today's date with the one which is inside the certificate;
- **Issuer search:** All certificates have an issuer; even the self-signed certificate has an issuer which is itself. At this phase, this process tries to find his issuer certificate which will be used afterwards;
- **Signature verification:** Each produced certificate contains in itself a signature which is produced by its issuer. This process consists on verifying this signature using the issuer certificate previously found;
- **Issuer verification:** Finally, the last phase of this module is to verify if the issuer is trustful or not; this means that the issuer must be a self-signed certificate where this process ends. If not, the issuer must pass the same process as the current certificate under verification.

4.1.4 Certificate Revocation Gateway

This module is responsible for verifying whether a certain certificate is revoked or not. In order to do that there are two mechanisms that can be used, which are OCSP and CRLs. In this case, priority is given to OCSP for several reasons. First of all, the OCSP repository is updated more frequently than the CRLs one, providing more accuracy. Furthermore, the OCSP is more efficient in a way that the response only contains information regarding the requested certification; on the other hand, CRLs contain information of several certificates, resulting in a higher overhead. However, CRLs are still used on this work; if the OSCP service is not available, the certificate revocation gateway redirects the request to the CRL verifier module.

Since this is a critical piece of software which deals with identity verification, in cases where it is not possible to verify whether the user's certificate is revoked or not (because the system has no Internet connection to execute an OCSP nor a CRL), the author chose to abort execution, informing the requester that it is not possible to verify the requested identity. Figure 13 illustrates the flow of verification on this module.

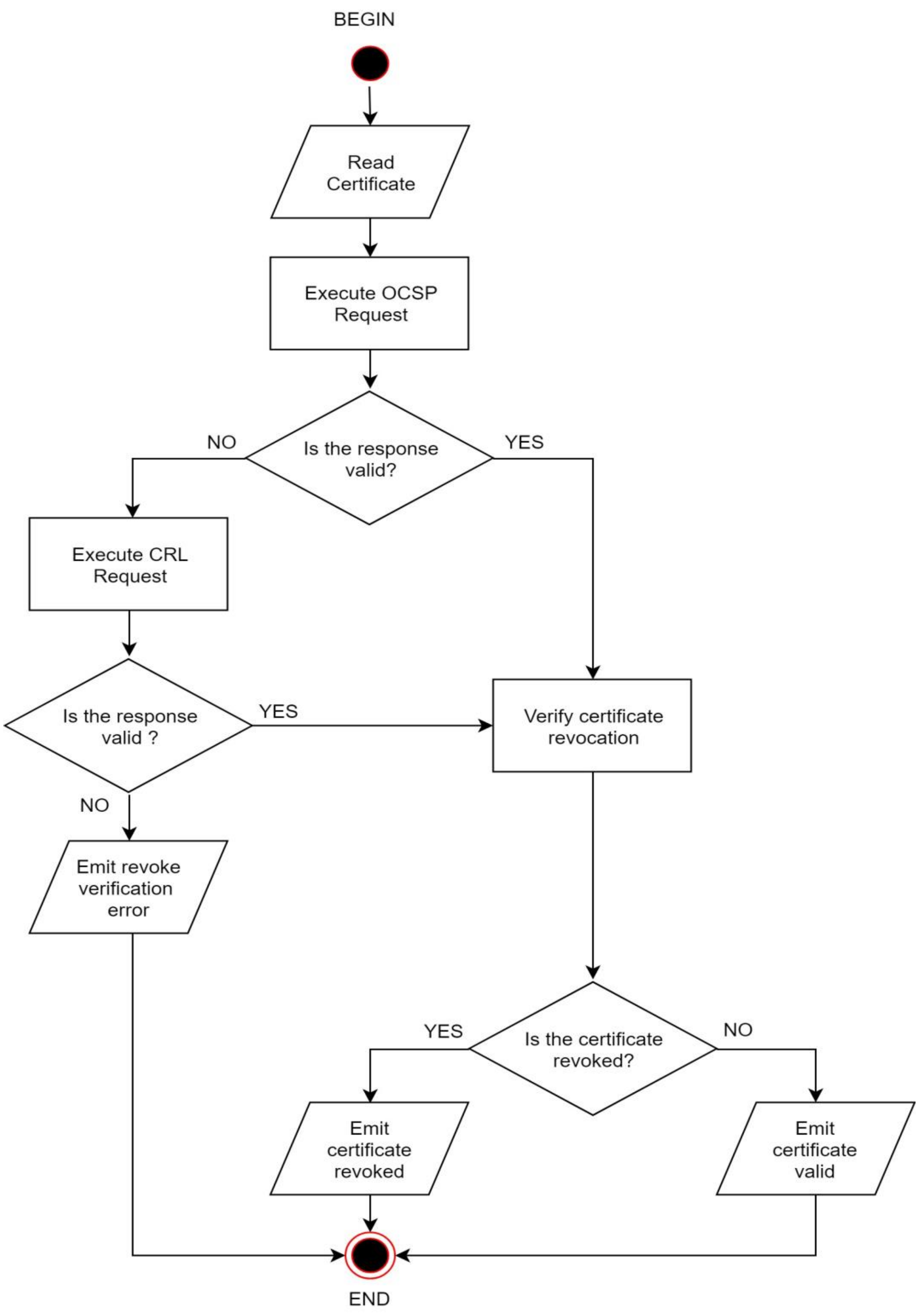
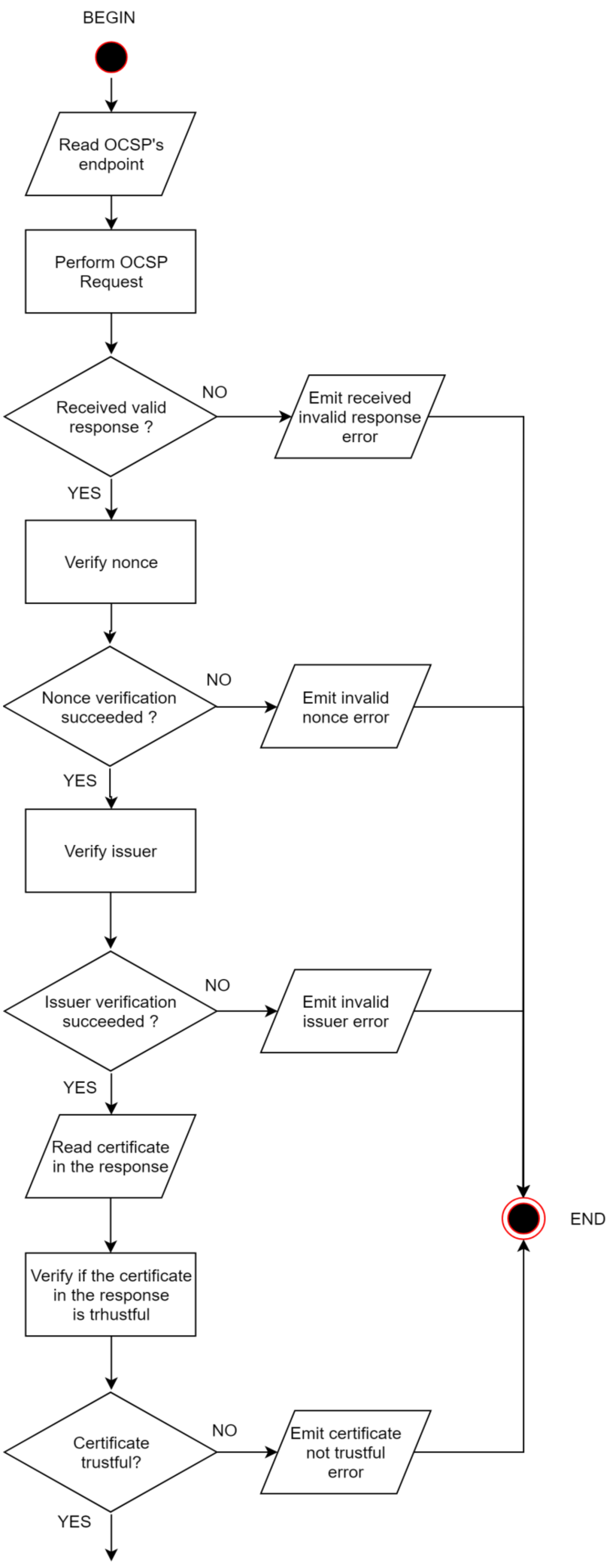


Figure 13 - Certificate Revocation Flow

4.1.5 OCSP Verifier

This module is responsible for asking to a certain Certification Authority (CA) if a certificate is revoked. In order to do that, there are several phases that must be accomplished. The flowchart on Figure 14 illustrates how this process is handled.


BEGIN
Read OCSP's endpoint
Perform OCSP Request
Received valid response ?
NO
Emit received invalid response error
YES
Verify nonce
Nonce verification succeeded ?
NO
Emit invalid nonce error
YES
Verify issuer
Issuer verification succeeded ?
NO
Emit invalid issuer error
YES
Read certificate in the response
END
Verify if the certificate in the response is trhustful
Certificate trustful?
NO
Emit certificate not trustful error
YES

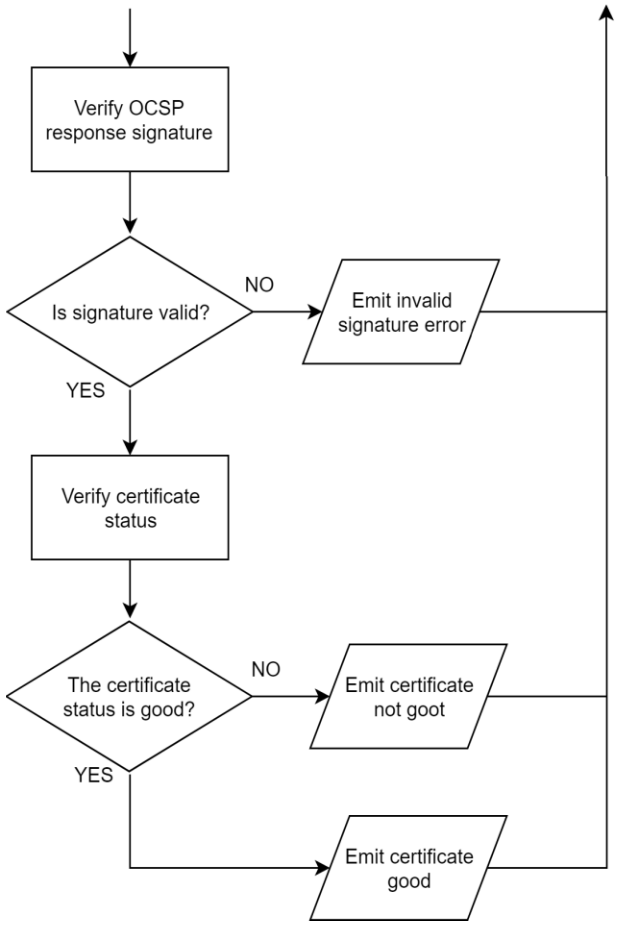

Verify OCSP response signature
Is signature valid?
NO
Emit invalid signature error
YES
Verify certificate status
The certificate status is good?
NO
Emit certificate not goot
YES
Emit certificate good

Figure 14 - OCSP Verifier flow

First of all, the OCSP address to be contacted must be obtained from the certificate. The process that involves the request creation has some particularities. For each request, the following information is added:

- **Certificate Identifier:** This identifier is the serial number of the certificate to be verified. The OCSP server will use this identifier to search for the certificate revocation status;
- **Issuer's Name Hash:** Is the hash of the issuer's Distinguished Name (DN);
- **Issuer's Key Hash:** Is the hash of the issuer's public key;
- **Hash Algorithm:** The hash algorithm used to create both hashes.
- **Nonce:** A nonce is being created and added to the request. The nonce will prevent replay attacks; in order to do that, the nonce binds the request to a certain response. This action is defined as an extension to the OCSP standard.

Once the request is performed, the server will verify the target certificate and return a response. In order to trust on the server response, some validations are performed:

- **Response validation:** First of all, it is verified if the response is valid; if so, the response status must have the successful state;
- **Nonce verification:** Then, the next verification to be performed is the nonce; as mentioned before, the nonce is very important since it avoids replay attacks;
- **Issuer verification:** On this step it is verified if the issuer data sent on the request is the same as the one received on the response;
- **Certificate verification:** In order to trust on the signature, the certificate must be verified as trustful. In order to do that, the verifier tries to build and verify a certification path; if that path is trustful, the certificate will be as well;
- **Signature verification:** The response is signed by the entity that takes the target certificate verification. This signature is verified with the certificate public key which is sent in the response;
- **Revoke verification:** There are three revocation status – *good*, *revoked* and *unknown*. If the received status is different from good, the verification is aborted, and an exception is emitted. In this case, the identity verification fails.

4.1.6 CRL Verifier

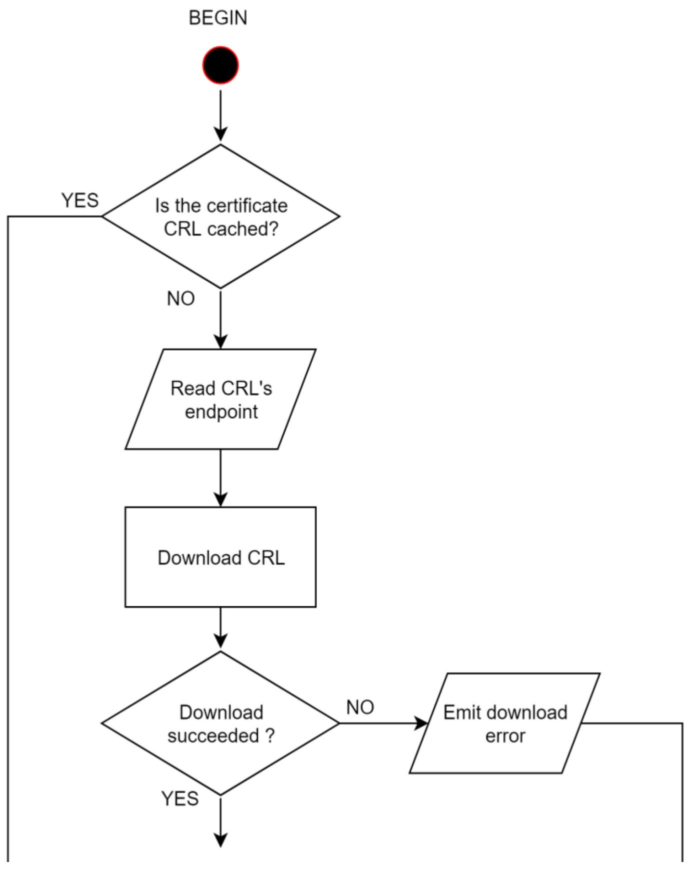
CRLs are only used when OCSP is not available. However, a new CRL is requested only in two cases:

- There is no CRL locally cached. Every time a new CRL is downloaded, that file is stored locally in order to be used as cache;
- In the case where the CRL is stored locally, it has to be checked for validity before using it to verify the certificate revocation status. CRLs contain the date of the next update to be performed; if the current date is ahead of the next update date, the current CRL is deleted and a new one is requested.

The following items will describe how this verifier works step-by-step:

- **Cache verification:** First of all, before executing a request in order to download the CRL, the download CRLs are cached locally. If the CRL is already stored and is still updated, this verifier does not need to request it again;
- **Download CRL:** If the CRL is not stored locally, the verifier will fetch the URL from the PTCC certificate and then it downloads it;
- **Chain verification:** Before keeping the CRL in cache and search for the certificate status on it, the trustfulness of the certificate which signs the CRL must be verified. This process is performed through a certificate verification as was performed with OCSP certificate;
- **Signature verification:** Then, if the certificate is trustful, the CRL signature will be verified with the certificate public key;
- **Revoke verification:** Once all these verifications are completed, the verifier searches for the certificate serial number on the CRL.

This flow is illustrated on Figure 15.



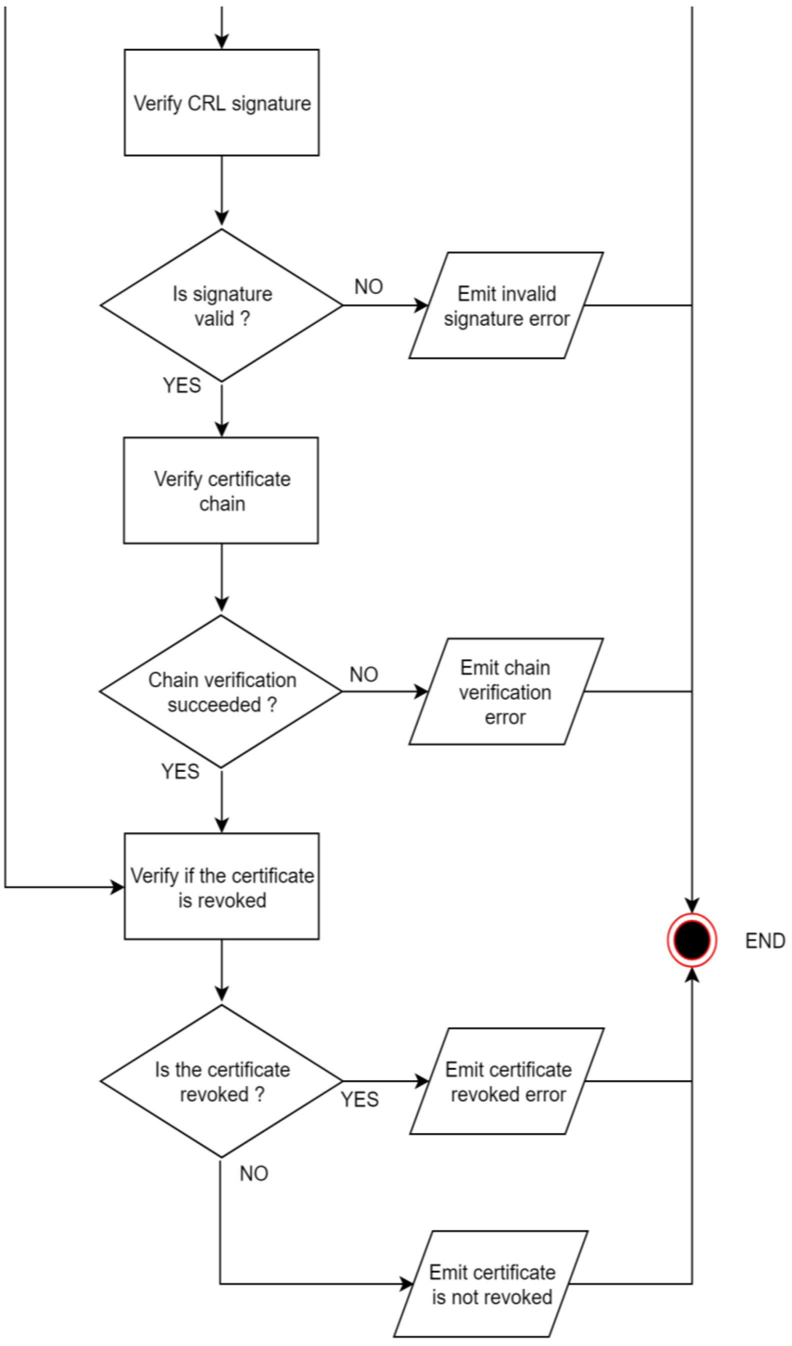

Verify CRL signature
Is signature valid ?
NO
Emit invalid signature error
YES
Verify certificate chain
Chain verification succeeded ?
NO
Emit chain verification error
YES
Verify if the certificate is revoked
END
Is the certificate revoked ?
YES
Emit certificate revoked error
NO
Emit certificate is not revoked

Figure 15 - CRL verifier flow

4.1.7 Wallet Address Verifier

Each WalliD user owns a MetaMask wallet which has an address. In order to verify if a certain wallet belongs to a user, the ImportiD signs with the PTCC private key the user’s wallet address. This signature is placed in hexadecimal format in the verifyID block that is stored in the StoreiD component. Then, this signature is verified through the

PTCC public key which is in the certificate that was verified by the certificate verifier component. This flow is illustrated on Figure 16.

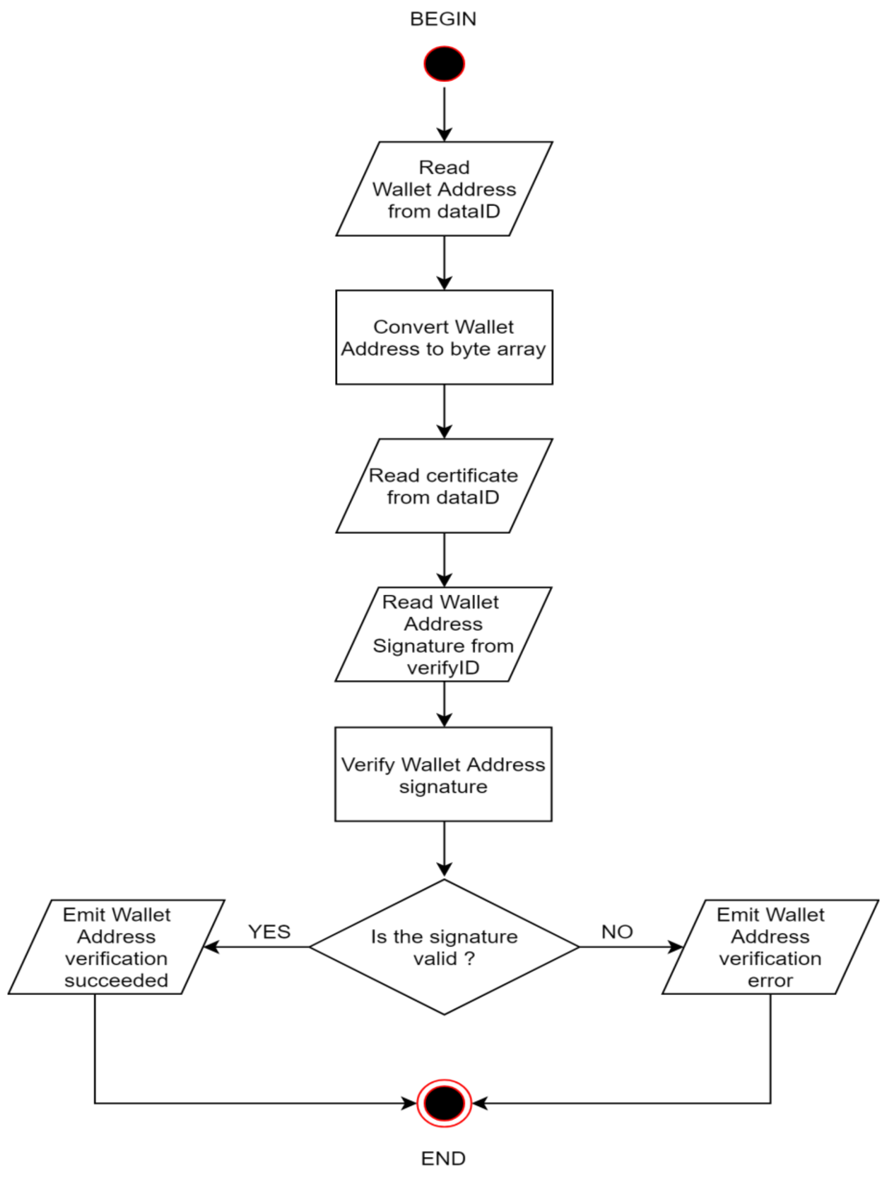

BEGIN
Read Wallet Address from dataID
Convert Wallet Address to byte array
Read certificate from dataID
Read Wallet Address Signature from verifyID
Verify Wallet Address signature
Is the signature valid ?
YES
NO
Emit Wallet Address verification succeeded
Emit Wallet Address verification error
END

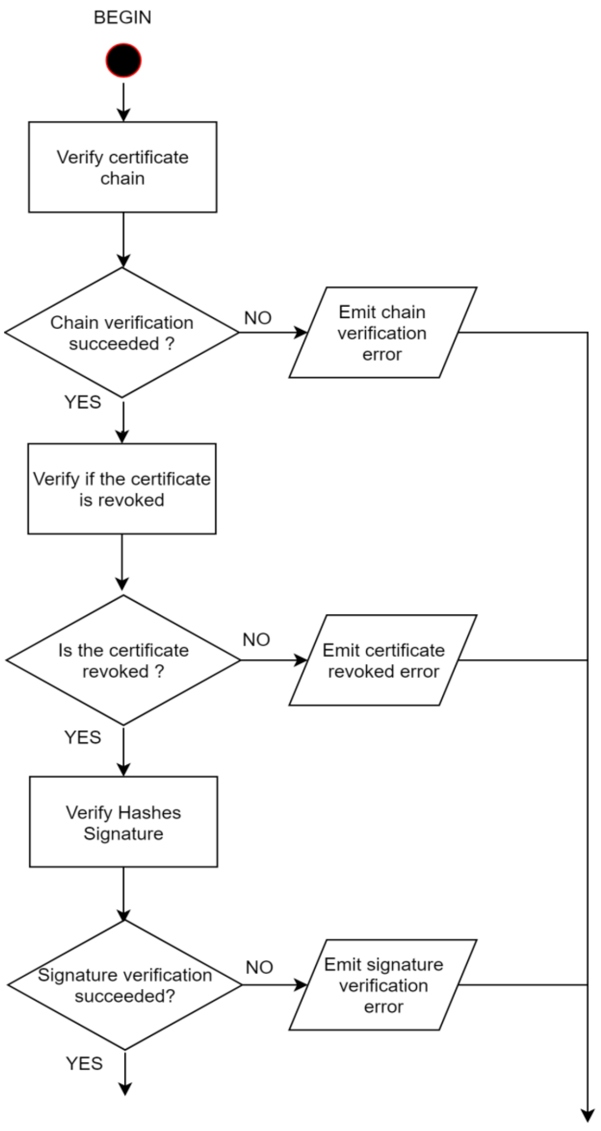
Figure 16 - Wallet Address Verifier

4.1.8 SOD Verifier

In order to make sure that the data stored in the PTCC is trustful, a file named Security Data Object (SOD) is stored in the card containing digital signatures of identity attributes, address, photo and the authentication public key. Each of this set of attributes were hashed using SHA-256 algorithm and then signed through the SOD certificate issuer. In order to trust these data, the next steps are followed:

- **Certificate verification:** First of all, as in the case of PTCC certificate, the SOD certificate must be performed as well in order to trust it.
- **Signature verification:** The verification hashes were signed by the SOD issuer. In order to trust them, this signature must be verified first.

From now on, the SOD file can be trusted. Since WalliD is only storing identity and address attributes, only these two hashes need to be verified. The process to verify them is very clear: for both cases, the attributes just need to be appended following a specific order. Then, a SHA-256 hash is generated for each of them. Once the hashes are generated, they are compared with the ones that are stored in SOD; if they are equal, then the attributes are trustful. This process is illustrated on Figure 17.


BEGIN
Verify certificate chain
Chain verification succeeded ?
NO
Emit chain verification error
YES
Verify if the certificate is revoked
Is the certificate revoked ?
NO
Emit certificate revoked error
YES
Verify Hashes Signature
Signature verification succeeded?
NO
Emit signature verification error
YES

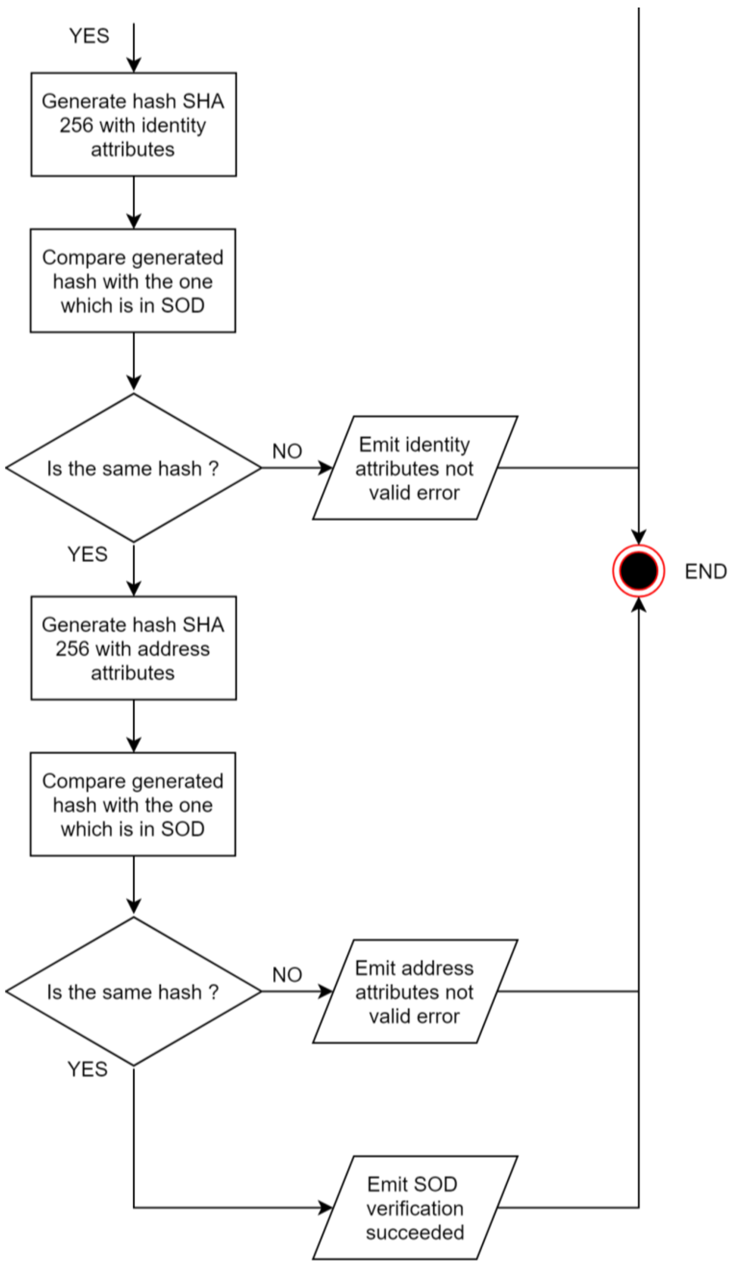

YES
Generate hash SHA 256 with identity attributes
Compare generated hash with the one which is in SOD
Is the same hash ?
NO
Emit identity attributes not valid error
YES
END
Generate hash SHA 256 with address attributes
Compare generated hash with the one which is in SOD
Is the same hash ?
NO
Emit address attributes not valid error
YES
Emit SOD verification succeeded

Figure 17 - SOD verifier flow

4.2 Integration

In order to easily integrate this KYC solution, a Java REST web service was conceived which receives the dataID block and processes it. The JAR file produced on this work is included as a library on this web service. Then, KYC service will be available through a POST with the route name `verify`. As a result, the web service will reply with a JSON which carries the operation result, which is `true` if that dataID block was verified successfully, and `false` if not. Furthermore, the response message will also contain a message reason carrying error reasons that are identified during the dataID block verification.

In order to verify users’ certificates which are within the Portuguese Citizenship Card, this library requires to access issuers’ certificates. Since they are several, a good way to centralize them all is to store them in a keystore. The path where this library is placed is passed to this library through an environment variable. The needed environment variables are:

- **KEYSTORE_PATH:** This variable defines the path where the keystore which stores the issuers’ certificates is placed;
- **KEYSTORE_PASSWORD:** This variable is used to store the password used to access the certificates stored in the keystore.

This KYC library has two operation modes: production and debugging. The production mode is enabled by default; on this mode, the library does not print any logs. On other hand, the debugging mode prints to the console and to a certain file. In order to enable the debugging mode, these two environment variables must be defined.

- **KYC_DEBUG:** This variable is used to enable the debugging mode. By default, the library assumes this variable as false; once turned to true, the library will start printing logs to the console.
- **KYC_DEBUG_FILE_PATH:** This variable provides to the developer the possibility to define a certain file path where the logs will be stored.

Then, in order to turn this solution easier to deploy, a docker image which contains this web service was created. Docker [42] is an open platform for developers and system administrators to build, ship, and run applications. These applications run in a docker container which is a controlled environment such as a virtual machine. A container [43] includes in itself all needed dependencies, system tools and libraries needed to run our applications. Figure 18 illustrates the architecture of how Docker runs our containers on the operation system.

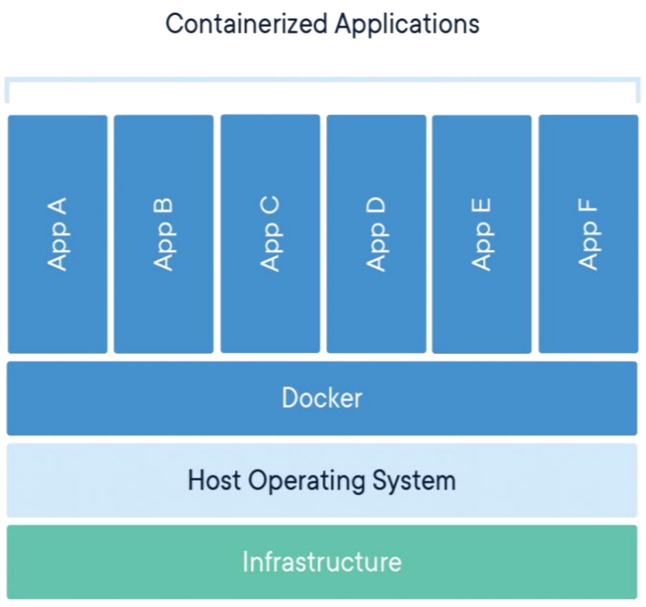


Figure 18 - Docker containers [43]

Docker makes deployment easier to be performed since it just requires a Docker image to be built, which will run in a Docker container. These images can be deployed everywhere since the environment has Docker installed. A Docker image is a set of layers which are built from Dockerfiles. A Dockerfile is a kind of recipe which contains lines of code that are used to create an environment where our services will be deployed.

4.3 Class Diagram

Figure 19 and Figure 20 present the class diagram of the solution presented on this work. Only Plain Old Java (POJO) and Guice classes were omitted in order to present only the most important classes of the system.

The mechanisms behind each class were already described on the previous sections.

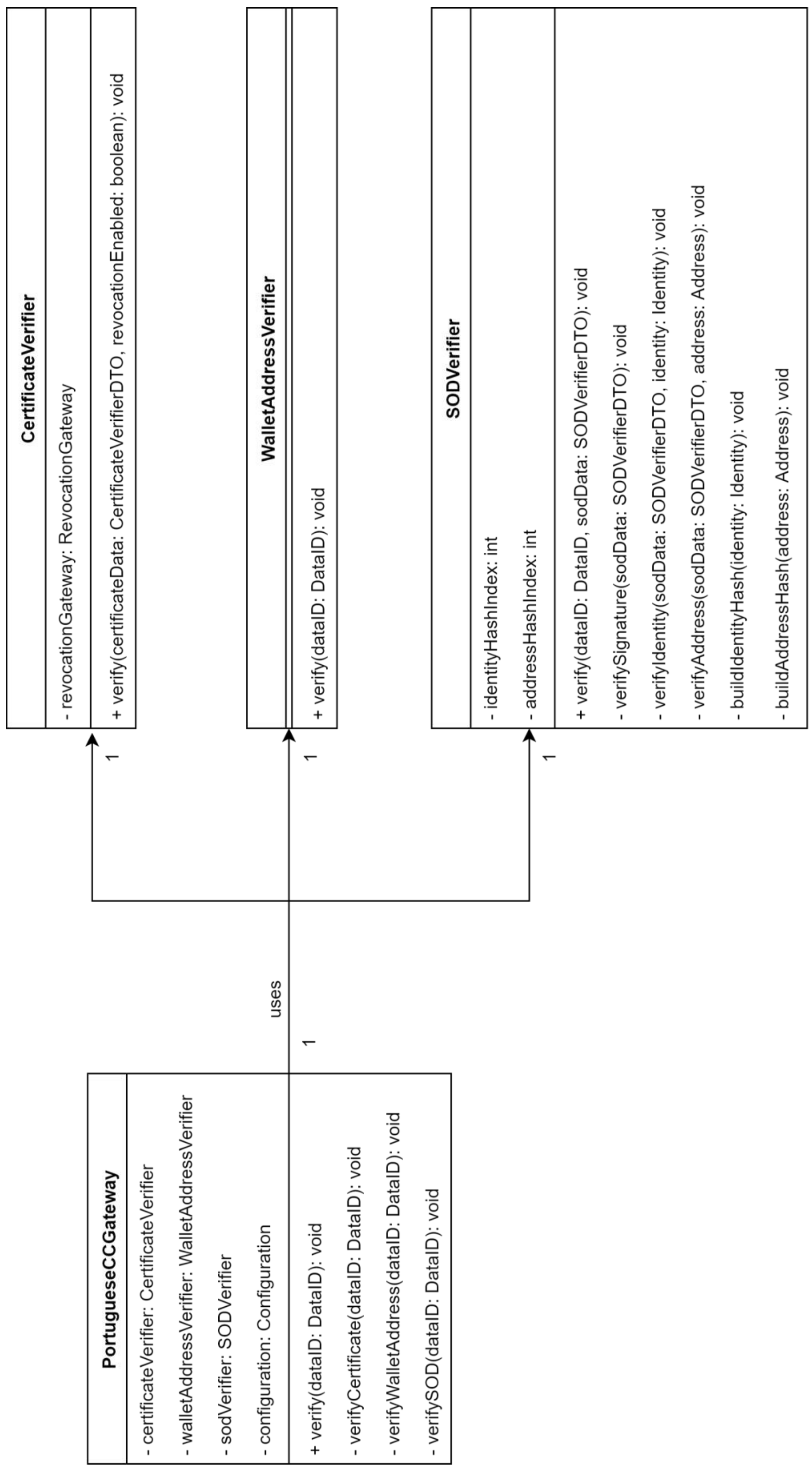

CertificateVerifier
- revocationGateway: RevocationGateway
+ verify(certificateData: CertificateVerifierDTO, revocationEnabled: boolean): void
WalletAddressVerifier
+ verify(dataID: DataID): void
SODVerifier
- identityHashIndex: int
- addressHashIndex: int
+ verify(dataID: DataID, sodData: SODVerifierDTO): void
- verifySignature(sodData: SODVerifierDTO): void
- verifyIdentity(sodData: SODVerifierDTO, identity: Identity): void
- verifyAddress(sodData: SODVerifierDTO, address: Address): void
- buildIdentityHash(identity: Identity): void
- buildAddressHash(address: Address): void
1
1
1
uses
1
PortugueseCCGateway
- certificateVerifier: CertificateVerifier
- walletAddressVerifier: WalletAddressVerifier
- sodVerifier: SODVerifier
- configuration: Configuration
+ verify(dataID: DataID): void
- verifyCertificate(dataID: DataID): void
- verifyWalletAddress(dataID: DataID): void
- verifySOD(dataID: DataID): void

Figure 19 - Class Diagram Part I

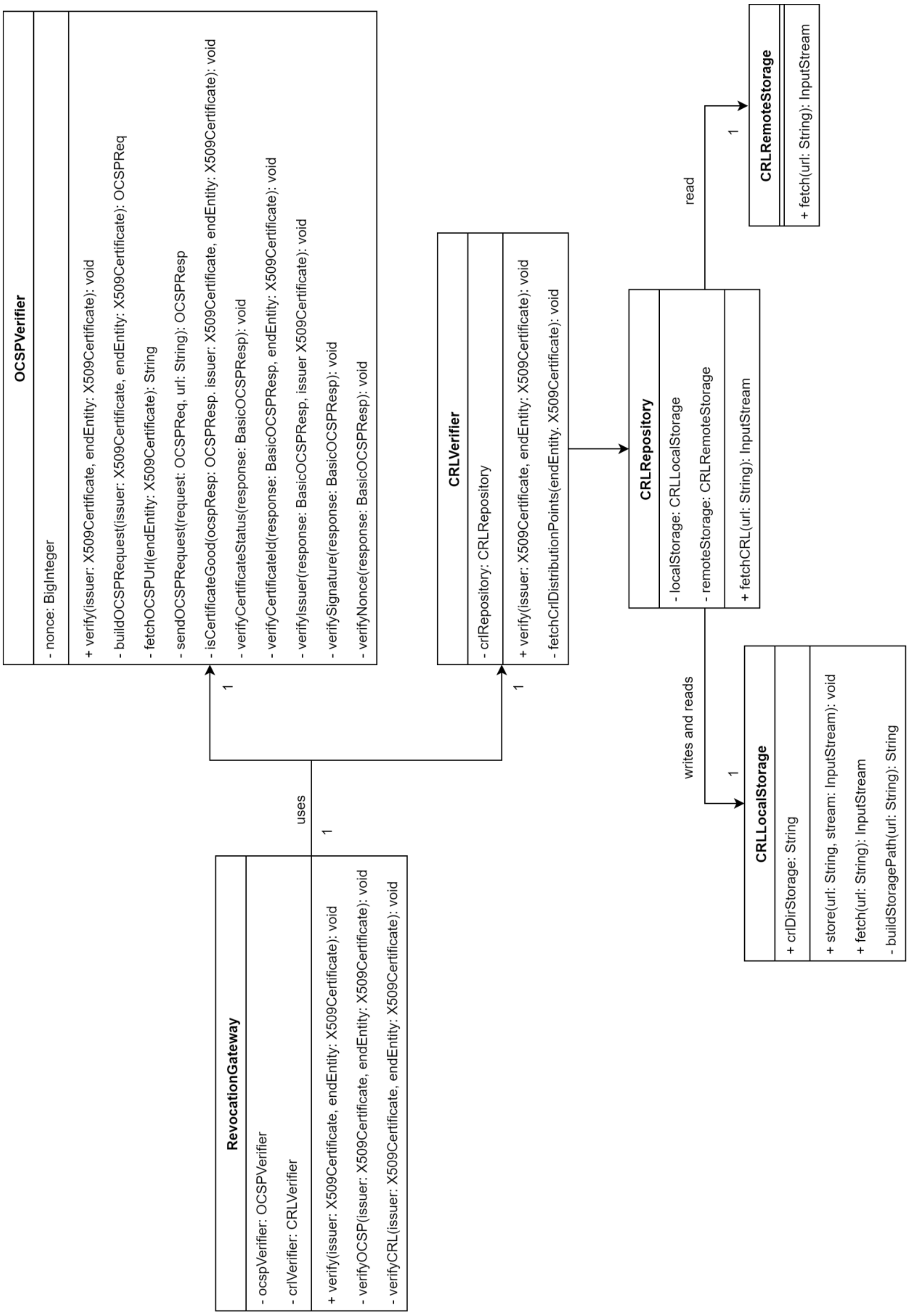

OCSPVerifier
- nonce: BigInteger
+ verify(issuer: X509Certificate, endEntity: X509Certificate): void
- buildOCSPRequest(issuer: X509Certificate, endEntity: X509Certificate): OCSPReq
- fetchOCSPUrl(endEntity: X509Certificate): String
- sendOCSPRequest(request: OCSPReq, url: String): OCSPResp
- isCertificateGood(ocspResp: OCSPResp, issuer: X509Certificate, endEntity: X509Certificate): void
- verifyCertificateStatus(response: BasicOCSPResp): void
- verifyCertificateId(response: BasicOCSPResp, endEntity: X509Certificate): void
- verifyIssuer(response: BasicOCSPResp, issuer X509Certificate): void
- verifySignature(response: BasicOCSPResp): void
- verifyNonce(response: BasicOCSPResp): void
CRLVerifier
- crlRepository: CRLRepository
+ verify(issuer: X509Certificate, endEntity: X509Certificate): void
- fetchCrlDistributionPoints(endEntity. X509Certificate): void
CRLRepository
- localStorage: CRLLocalStorage
- remoteStorage: CRLRemoteStorage
+ fetchCRL(url: String): InputStream
CRLRemoteStorage
+ fetch(url: String): InputStream
CRLLocalStorage
+ crlDirStorage: String
+ store(url: String, stream: InputStream): void
+ fetch(url: String): InputStream
- buildStoragePath(url: String): String
RevocationGateway
- ocspVerifier: OCSPVerifier
- crlVerifier: CRLVerifier
+ verify(issuer: X509Certificate, endEntity: X509Certificate): void
- verifyOCSP(issuer: X509Certificate, endEntity: X509Certificate): void
- verifyCRL(issuer: X509Certificate, endEntity: X509Certificate): void
uses
1
1
1
read
1
writes and reads
1

Figure 20 - Class Diagram Part II

Chapter 5

Evaluation

This chapter presents how the proposed KYC library reacts to a set of test cases. In order to perform this explanation, let us take in consideration Credibank's use-cases. For the following scenarios, the dataID block will be omitted for privacy meanings, as these tests were conducted using some real Portuguese Citizenship Cards.

The chapter is structured as follows: Section 5.1 describes how the Application Programming Interface (API) for each test case; Section 5.2 presents some performance tests performed through the API.

5.1 API

As mentioned before, the developed library receives as input a dataID block which is processed and then returns a result. On this case, the web service that is using the library provides the results in a JSON format. This JSON has only two keys:

- **verificationSuccessful:** This key will contain the true value if the user's identity was successfully verified. If any error occurs during this process, the returned value will be `false`;
- **reason:** This key returns a string containing the cause of the failure during the identity verification process.

Since Credibank is a credit institution that aims to provide loans to their customers, it makes sense that they need to validate the customer's identity. The first test case to be presented is a positive scenario; this means that the user's identity is verified successfully, without any errors. Figure 21 presents the related response.

```
{
    "verificationSuccessful": true,
    "reason": null
}
```

Figure 21 - Positive Response

Figure 22 presents a response of an identity which has the certificate revoked. This could happen for example if a citizen loses his Portuguese Citizenship Card and asks for a new one.

```
{
    "verificationSuccessful": false,
    "reason": "Certificate status is not good"
}
```

Figure 22 - Revoked Certificate Response

Figure 23 illustrates a response of an identity which contains an expired certificate. This scenario can occur if a certain Portuguese Citizenship Card gets expired.

```
{
    "verificationSuccessful": false,
    "reason": "NotAfter: Tue Feb 27 00:00:00 UTC 2018"
}
```

Figure 23 - Expired Certificate

Figure 24 presents a scenario where the wallet address and its signature do not match.

```
{
    "verificationSuccessful": false,
    "reason": "Wallet Address signature verification
failed."
}
```

Figure 24 - Wallet Signature verification failed

The response presented on Figure 25 represents a scenario where there is an error during the user's identity attributes verification process. This message will be triggered for example if a user tries to forge his own identity attributes, like changing his name.

```
{
    "verificationSuccessful": false,
    "reason": "Identity verification failed."
}
```

Figure 25 - Identity attributes verification failed

A scenario where a user that tries to forge his address attributes, such as street name, zip code, the software will detect that inconsistency and will return the response illustrated on Figure 26.

```
{
    "verificationSuccessful": false,
    "reason": "Address verification failed."
}
```

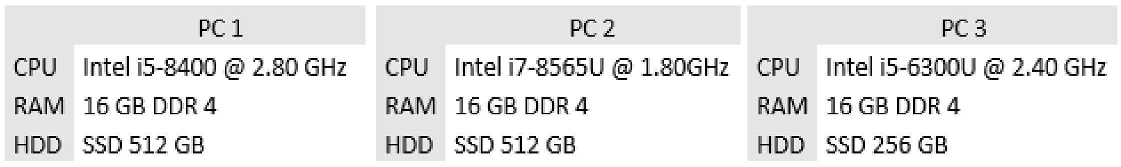
Figure 26 - Address attributes verification failed

As presented on this section, the API seems to be very concise and clear on its outputs. For each failure case, seems to be pretty clear and concise what each of these reasons are meant to transmit.

5.2 Performance Tests

In order to present how this library reacts in load scenarios, some tests were performed to simulate this kind of situations.

The following tests were performed five times each, the values presented are an average of the collected values. These results can suffer changes depending of the hardware, Internet connection and the current load of the remote servers. The hardware used on these tests are depicted on Figure 27.



	PC 1		PC 2		PC 3
CPU	Intel i5-8400 @ 2.80 GHz	CPU	Intel i7-8565U @ 1.80GHz	CPU	Intel i5-6300U @ 2.40 GHz
RAM	16 GB DDR 4	RAM	16 GB DDR 4	RAM	16 GB DDR 4
HDD	SSD 512 GB	HDD	SSD 512 GB	HDD	SSD 256 GB

Figure 27 - Hardware used during the performance tests

The generated loads were performed by PC 3 under three scenarios which are: 20 threads where each of them will produce 100 requests, 80 threads where each of them produced 100 requests and so on. The following requests where produced through three different Portuguese Citizenship Cards. The usage of these documents was completely random. The chart presented on Figure 28 presents how the produced library reacts on PC 1 to a certain load.

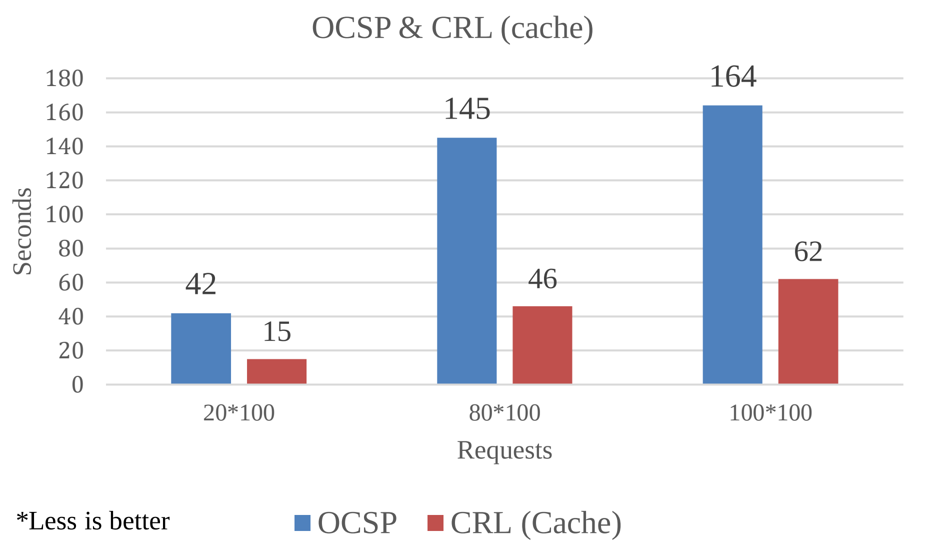


Figure 28 - System performance with PC 1

The first impression analyzing this chart is that CRL provides a better performance than OCSP. When a CRL is downloaded, the provider adds a tag saying until when that CRL will take the next update, in that case, the library is keeping that CRL for future requests, that is how the cache mechanism is implemented. The library could do the same for OCSP, in other words, the library could store the serial number and attach to it the last result retrieved. But this solution has a problem, the OCSP is updated much more frequently than the CRL. This CRLs are being updated one time per week, on the other hand, OCSP responses have no information of when the next update will take place. In that case, it is risky to cache this kind of data. In order to show that the cache mechanism is doing its job, take in consideration the chart presented on Figure 29.

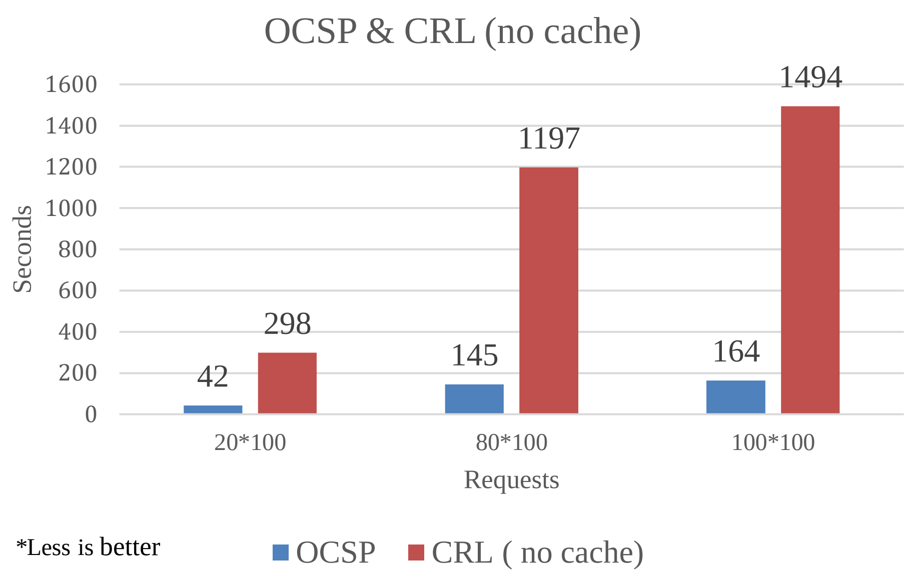


Figure 29 – System performance with PC 1 without cache

As illustrated on the previous chart, without cache the performance of CRL is far worse than OCSP. This occurs because CRLs do not contain only the requested certificate revocation information, but a whole range of them. In that case, the time required to download and search increases, which will result in a worse performance. Finally, but not less important, the chart illustrated on Figure 30 presents how the performance is scaling when the number of computers increase.

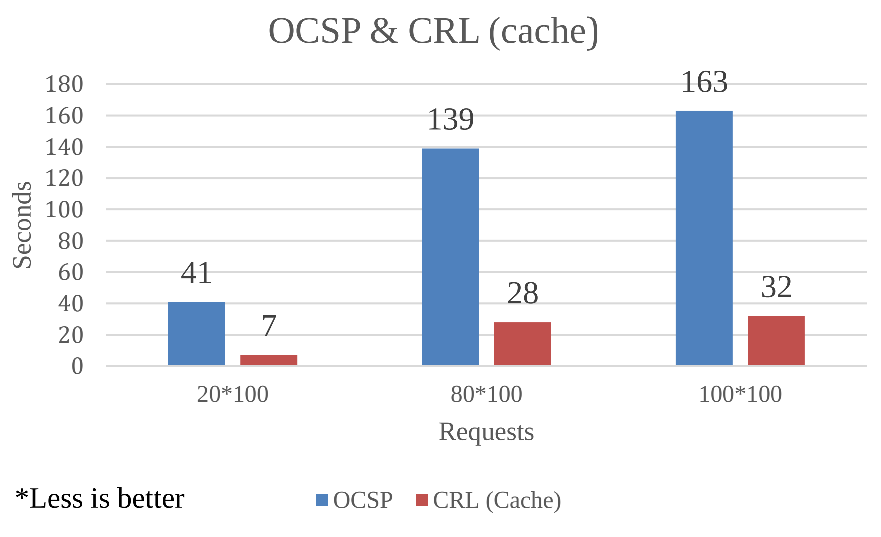


Figure 30 – System performance with PCs 1 and 2

For the OCSP point of view, the performance has been kept. No reason was found for this fact, maybe there is a kind of limitation on the OCSP service, since there are two instances dispatching the same kind of requests, the time spent to process the same request should decrease. On the other hand, it is possible to observe that behavior on CRL. Since the library is using a local cache, if the number of instances increase, the time spent to process those requests will decrease as well.

5.3 Summary

Using the referred examples and use-cases, it is possible to infer that the proposed solution, as an example of an application of blockchain, allows securing sensitive data in a safe and cyphered manner, whether directly on the blockchain or in a separate server which can be accessed with hash keys being stored in the blockchain. This verifies hypothesis HYP1. The blockchain in itself does not mandate that there will be solely one instance of the identification documents, as this is more about the policies regarding storage of data. What can be said is that the blockchain does provide a mechanism that allows a user to give a public key and address that allows multiple entities to access that information, while registering each access request, which is sufficient to verify hypothesis HYP2. The digital certificate within the SOD file aims to verify users' identity attributes, such as name, surname, address and so on. This approach is far better than ink on a piece of paper which needs to be verified by a notary, that even could take some risks. Furthermore, there is no need for a third party to verify a certain identity – this seems sufficient to verify hypothesis HYP3. Finally, but not less important, this work proves that it is possible to develop automatic KYC mechanisms, which could be used to tackle human errors, waste of money and speed up the legacy KYC mechanisms. This seems sufficient to verify the hypothesis HYP4.

Additionally, as presented on the previous section, it is possible as well to give some feedback related with a possible failure during the identity verification process, which could be useful too. Furthermore, it is possible to provide this feature as a service, which in that case, will allow companies to remove third parties from identity verification processes. There are more advantages on this kind of services: the minimization of human error during these processes, the speed up of these and the cost of these processes can be very appreciated for some entities that aim to use this kind of services.

Chapter 6

Conclusions & Future Work

Nowadays, performing a proof of identity in the online world is a complex issue. The current solutions do not provide the same trust as we have in the physical world. For instance, the CIVIC project assumes that the user is not acting in misbehavior, in other words, it trusts that the user will really insert his identity attributes. Some KYC solutions are human based, which may cause errors and slow down these processes.

WalliD uses trust on the entities which issue our identity cards on the real world. Those identity attributes can be validated anytime through its digital certificate, and this brings the confidence that other solutions do not ensure. Furthermore, WalliD also provides a way to any entity to build their own KYC system – this work will prove that these processes can be performed in a fast and secure way.

As described in the previous chapter, WalliD is an example of an application that can satisfy the research questions proposed in this work.

Nevertheless, WalliD does have a critical issue that is related with a statement of General Data Protection Regulation (GDPR) which is “the right to be forgotten”. In the case the data is stored in the blockchain, although ciphered, there is no way to remove it. WalliD also assumes that the user’s identity contains a digital certificate to verify the user’s identity attributes. Similarly, KYC process also does this assumption.

Future work in this area includes integrating more KYC services in this library which will help WalliD to grow up, such as Spanish citizenship card which uses X.509 certificates as well. The Portuguese government already announced the digital driver’s license, and this could be very interesting for WalliD. It could be possible to develop KYC services which may help insurance companies or rent-a-car companies to identify and verify their users’ identities automatically.

Publishing this library on the Maven Central Repository [44] would be a significant improvement. Maven Central Repository is a place where the developers can publish their libraries. Once they are published, any developer can use it freely under the software license attached to that software. In this way, any developer will be able to import this library as a Maven dependency, which turns the process of dependency management easier to perform.

As soon as the integration with Docusign finishes, the produced library will be released for the open source community. It is a goal as well that this library can get some improvements provided by the community, or even be audited by anyone that aims to use it as well.

In addition to this research work, the author also performed a quite extent set of contributions to the scientific community, such as:

The participation in numerous research projects, namely:

- H2020 vf-OS (Virtual Factory Operating System)
  - http://vf-os.eu
  - Consortium of Caixa Mágica Software with ICE (L), Ascora, Almende, Uninova, UPV, IKERLAN, Mondragón Assembly, Univ. Lyon 2, Via Solis, Consulgal, Knowledgebiz, APR and Tardy;
  - Funding: H2020-FoF-11-2016 – Digital Automation;
  - Project Nr. 723710 – Budget ≈ 8 M€;
  - Project duration: 2016-10-01 to 2019-09-30;
  - vf-OS aims to develop an Open Operating System for Virtual Factories, composed by a Virtual Factory System Kernel, a Virtual Factory Application Programming Interface and a Virtual Factory Middleware specifically designed for the factory of the future. An Open Applications Development Kit will be provided to software developers for deploying Manufacturing Smart Applications for industrial users, using the Manufacturing Applications Store at the Virtual Factory Platform. The Virtual Factory Platform will provide a range of services to the connected factory of the future to integrate better manufacturing and logistics processes;
  - Responsibilities: Main developer for partner Caixa Mágica Software, which participates in all the phases of the development, mainly the vf-OS Open Development SDK, Studio and Development Engagement Hub.

- H2020 UMOBILE
  - http://www.umobile-project.eu/
  - Consortium of Athena RC, UCL, UCAM, COPElabs, Tecnalia, Tekever, Senception, FON and AFA Systems;
  - Funding: H2020-ICT-05-2014 – Smart Networks and novel Internet Architectures;
  - Project Nr. 645124 – Budget ≈ 3 M€;
  - Project duration: 2015-02-01 to 2018-04-30;
  - UMOBILE aims to develop a communication system for emergency situations, such as fire, earthquake and so on. In order to do that, a universal mobile-centric and opportunistic communications architecture was developed, integrating the principles of delay tolerant networks. Then, the people which were facing disaster scenarios could use their smartphones to generate messages, such as, indicating that a certain road is destroyed or doomed by fire. Those messages were transferred through Wi-Fi Direct device to device, that could be smartphones nearby or drones that were used to carry those messages far away;
  - Responsibilities: Main developer for partner COPElabs, developing an Android service which deals with the messaging exchange generated by smartphones. This service implements a routing algorithm developed during the project, which aims to choose the best next hop to transfer a certain message.

The elaboration and writing of papers published in the proceedings of International Conferences:

- M. Tavares, F. Veiga, A. Guerreiro, A. Campos, C. Coutinho, “WalliD: Secure your ID in an Ethereum Wallet”, in *Proceedings of the 9th IEEE-TEMS international Conference on Intelligent Systems*, Madeira, Portugal, 2018;
- M. Tavares, O. Aponte, P. Mendes, "Named-data Emergency Network Services", in *ACM Mobisys 16th Annual International Conference on Mobile Systems, Applications, and Services*, Munich, Germany, 2018.

Additionally, a paper was submitted to a special issue on “Instrumentation and Measurement context: From a methodological point of view” of the IEEE Instrumentation and Measurement Magazine[2], waiting for approval.

[2] https://ieeexplore.ieee.org/xpl/aboutJournal.jsp?punumber=5289

Glossary

AI	Artificial Intelligence
API	Application Programming Interface
CRLs	Certificate Revocation Lists
CA	Certification Authority
CMS	Cryptographic Message Syntax
DI	Dependency Injection
DER	Distinguished Encoding Rules
DN	Distinguished Name
EE	End Entity
GDPR	General Data Protection Regulation
IDT	Identity Type
KYC	Know Your Customer
JAR	Java Archive
MAC	Message Authentication Code
OCL	Object Constraint Language
OCSP	Online Certificate Status Protocol
OPID	Operation Unique Identifier
PAN	Permanent Account Number
PC	Personal Computer
PBFT	Practical Byzantine Fault Tolerance
PEM	Privacy-enhanced Electronic Mail
PoI	Proof of Identity
PoW	Proof of Work
POJO	Plain Old Java Object
PKIX	Public Key Infrastructure for X.509
PTCC	Portuguese Citizenship Card
RA	Registration Authority
SDKEY	StoreiD Session Public Key
TDD	Test-Driven Development
UML	Unified Modelling Language

WA	Wallet Address

Bibliography

[1] "WalliD", [Online]. Available: https://wallid.io [Accessed 18-11-2018].

[2] "Recommendation X.509", ITU - International Telecommunication Union, [Online]. Available: http://www.itu.int/rec/T-REC-X.509. [Accessed 10-01-2018].

[3] R. Mui, "Know Your Customer compliance costs continue to grow at financial firms: surveys", 27-10-2018. [Online]. Available: https://www.businesstimes.com.sg/banking-finance/know-yourcustomer-compliance-costs-continue-to-grow-at-financial-firmssurveys. [Accessed 10- 01-2018].

[4] "Credibank", WalliD, [Online]. Available: http://credibank.herokuapp.com. [Accessed 18-11-2018].

[5] M. Tavares, F. Veiga, A. Guerreiro, A. Campos, C. Coutinho, “WalliD: Secure your ID in an Ethereum Wallet”, in *Proceedings of the 9th IEEE-TEMS international Conference on Intelligent Systems*, Madeira, Portugal, 2018.

[6] “Docusign”, [Online]. Available: https://www.docusign.com. [Accessed 10-04-2019].

[7] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System", 2009.

[8] M. Castro, B. Liskov, “Practical Byzantine Fault Tolerance”, in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, New Orleans, USA, February 1999.

[9] "Ethereum", [Online]. Available: https://www.ethereum.org. [Accessed 18-11-2018].

[10] “Ethereum White Paper”, [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper. [Accessed 12-02-2019].

[11] H. Syahputra, H. Weigand, "The Development of Smart Contracts for Heterogeneous Blockchains", in *Interoperability for Enterprises Systems and Applications (I-ESA 2018)*, Berlin, Germany, 2018.

[12] "Civic Secure Identity Ecosystem", [Online]. Available: https://www.civic.com. [Accessed 24-04-2018].

[13] "uPort: Open Identity System for the Decentralized Web", [Online]. Available: https://www.uport.me. [Accessed 04-05-2018].

[14] "Persona: Digital Identity Management System on Blockchain", [Online]. Available: https://persona.im. [Accessed 20-03-2018].

[15] D. Duccini, "IDCoins: A Web of Trust Blockchain for Identity and Reputation", 2014 [Online]. Available: https://github.com/IDCoin/IDCoin. [Accessed 21-06-2019].

[16] "Selfkey: Your key to Freedom", [Online]. Available: https://selfkey.org. [Accessed 26-04-2018].

[17] "TheKey: Decentralized Ecosystem of an Identity Verification Tool", [Online]. Available: https://www.thekey.vip. [Accessed 04-05-2018].

[18] "REMME Portal", [Online]. Available: https://remme.io. [Accessed 27-06-2018].

[19] "Guardtime Portal", [Online]. Available: https://guardtime.com. [Accessed 07-07-2018].

[20] "Bitnation Portal", [Online]. Available: https://bitnation.co. [Accessed 20-07-2018].

[21] “Jumio”, [Online]. Available: https://www.jumio.com/. [Accessed 02-01-2019].

[22] “Shufti Pro”, [Online]. Available: https://shuftipro.com/. [Accessed 02-01-2019].

[23] P. Mondal, R. Deb, M. Huda, “Know Your Customer (KYC) based authentication method for financial services through the internet”, in *Proceeding of the 19th International Conference on Computer and Information Technology*, North South University, Dhaka -1229, Bangladesh, 2016.

[24] A. Soni, R. Duggal, “Reducing Risk in KYC (Know Your Customer) for large Indian banks using Big Data Analytics”, *International Journal of Computer Applications,* vol. 97, pp. 49-53, 2014.

[25] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile RFC 5280", Internet Engineering Task Force (IETF), 2008.

[26] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams, "X.509 Internet Public Key Infrastructure RFC6960", Internet Engineering Task Force (IETF), 2013.

[27] R. Housley, “Cryptographic Message Syntax (CMS) 5652”, Internet Engineering Task Force (IETF), 2009.

[28] B. Kaliski, “PKCS #7: Cryptographic Message Syntax Version 1.5”, Internet Engineering Task Force (IETF), 1998.

[29] E. Moriarty. et al., “PKCS #12: Personal Information Exchange Syntax v1.1”, Internet Engineering Task Force (IETF), 2014.

[30] "WalliD White Paper", [Online]. Available: https://wallid.io/docs/one-pager-v4.pdf. [Accessed 18-11-2018].

[31] "WalliD GitHub", [Online]. Available: https://github.com/walliDprotocol. [Accessed 18-11-2018].

[32] "Metamask", [Online]. Available: https://metamask.io. [Accessed 18-11-2018].

[33] "ImportiD GitHub", WalliD, [Online]. Available: https://github.com/walliDprotocol/wallid-ImportID. [Accessed 18-11-2018].

[34] "Manual técnico do Middleware Cartão de Cidadão", 2016, p. 24.

[35] "StoreiD", WalliD, [Online]. Available: https://github.com/walliDprotocol/wallid-StoreID. [Accessed 18-11-2018].

[36] “MyEtheriD”, WalliD, [Online]. Available: https://myetherid.io. [Accessed 18-11-2018].

[37] "MyEtheriD GitHub", WalliD, [Online]. Available: https://github.com/walliDprotocol/wallid-MyEtherID. [Accessed 18-11-2018].

[38] "Bouncy Castle", [Online]. Available: https://www.bouncycastle.org. [Accessed 18-11-2018].

[39] "Chave Movel Digital", [Online]. Available: https://www.autenticacao.gov.pt/a-chave-movel-digital. [Accessed 18-11-2018].

[40] "Guice", Google, [Online]. Available: https://github.com/google/guice. [Accessed 10-04-2019].

[41] “JUnit”, [Online]. Available: https://junit.org/junit5. [Accessed 18-11-2018].

[42] “Docker”, Docker, [Online]. Available: https://www.docker.com. [Accessed 03-04-2019].

[43] “Docker Containers”, Docker, [Online]. Available: https://www.docker.com/resources/what-container. [Accessed 03-04-2019].

[44] “Maven Central Repository”, Apache Maven Project, [Online]. Available: https://maven.apache.org/repository/. [Accessed 15-05-2019].