Designing the future identity: authentication and authorization through self-sovereign identity

Valentin Gerard

Delft University of Technology

Designing the future identity: authentication and authorization through self-sovereign identity

Master's Thesis in Computer Science
EIT Digital MSc Programme
Cloud Computing and Services

Parallel and Distributed Systems group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Valentin Gerard

21st August 2019

Author
Valentin Gerard

Title
Designing the future identity: authentication and authorization through self-sovereign identity

MSc presentation
30th August 2019

Abstract

The apparition of the Internet was a revolution that allowed to connect people all around the world. It was a disruptive technology that helped to shape the modern society as we know it today. However, to trust a person we interact with but we are not able to see is difficult. In order to trust people in this huge network, different digital identity models have been built to attempt to authenticate its users. Still, as the technology and our use of it evolved, the complexity of digital identity increased and problems started to appear. The two main problems are the aggregation of personal information in the databases of powerful IT companies and the user experience that is more complicated than it should be. Those two problems are linked and are caused by identity models that were not designed in a user-centric approach. Each organization uses its own identity system to authenticate their users, which causes the fragmentation of the user's digital identity that have to register and create identity for all the organizations he is interacting with.

Self-sovereign identity is an emerging identity model that makes use of distributed ledger technologies and cryptography to solve these problems. The goal of this model is to return digital identities in the hand of users by placing them at the center of the model. Unifying all their personal information under one identity would greatly improve their experience and help them to have a better control on how it is used by the organizations they're interacting with. In our first research question, we first examine if the self-sovereign architecture can provide an identity that can be trusted by people and organizations alike. We found that the trust that we can accord to this model is highly related to our capacity to safely manage the different cryptographic keys that are used to secure wallets where user's information is stored and to secure the communication channels between the different identity owners. The storage of these keys is the subject of our second research question. To understand better the mechanisms, we implement a prototype using the Hyperledger Indy blockchain to discover how these keys and the credentials associated to the digital identity are stored and managed in this ecosystem. We then use the prototype to authenticate a user and make use of its credentials to authorize or deny the access to specific resources on the system to make the link with identity and access management in our third research question.

Preface

Since my first encounter with the Internet I have been fascinated by the fact that you can instantly get almost any information from all over the world from anywhere as long as you are connected to the network. After high school, I started a general training in web professions where I learned mostly about the surface or the front-end as we call it in the field. To get the big picture I understood that I had to study computer science more in depth. That's how I ended up enrolling for the EIT Digital MSc Programme "Cloud Computing and Services" master, which as a bonus, gave me the possibility to travel and meet many people as well as opening the door to new opportunities. There, one specific subject got my interest. The simple idea that we can use and share a huge amount of resources through the very same network I discovered years ago to solve the most complex problems made me passionate about distributed systems. As a result, the complex problem I decided to work on for this thesis is digital identity. In my point of view, this subject is central to many others and will unlock new opportunities to link physical and digital worlds for a better future as long as it is designed for individuals first.

I was hired as an intern in 'Orange Application for Business', a French company with a cybersecurity department to work on this subject. My goal was to explore the digital identity subject and to make a link with recent advancements in blockchain technology to design a service we could build on it. That's how I started designing a self-sovereign identity proof-of-concept where is it possible to hold, issue and verify credentials in a secure way and where users' privacy is respected.

I would like to thank my colleagues from Orange Application for Business for welcoming me in their office and giving me the opportunity to work on this project. I would also like to thank my professors at TU Delft for their teaching. I extend my gratitude to my family which is with me since the very beginning, my oldest friends as well as the ones I met on my journey, the teachers which passed on their knowledge to me and to many other students.

Valentin Gerard

Rennes, France
21st August 2019

Contents

Chapter 1

Introduction

Trust has always been an important part of human interactions. When communities grew up and could no longer rely on their proximity[1], men invented ways to provide trust in larger ecosystems where most of the people do not know each other. It has often been done through the issuance of certificates on which we added seals to authenticate their issuer and used encryption methods to ensure the message could not be read by anyone. It did not change much since. However, to keep up with the Internet revolution, techniques had to evolve and they are now much more complex than before[2], although we are still using paper and plastic cards in the physical world. We are now using cryptography to answer our need to provide trust in our online interactions.

When you think about it, credentials are everywhere. We get our first one, the birth certificate, very early and we are collecting more of them all along our life. However, as stated before, they are mostly issued in a paper format or by using plastic cards and we have no standard way to assert their authenticity online. Or to be more accurate, no good way to do it. In existing architecture, we have to rely on third parties to keep our identities on our behalf. The biggest issue with this model is that our identities are locked in the service provider server and we have no way to reuse them for other services. We have to create a fragment of our identities for each service that we have to protect with multiple passwords. This leads to security and privacy issues as all these fragments become possible target for attackers.

In the last 20 years we started to see multiple initiatives to build a unique and persistent user-centric identity that would allow the secure exchange of credentials online like the Augmented Social Network[3] or more recently OpenID[4]. The most recent concept is called self-sovereign identity. In addition to a user-centered design, this concept defines an ecosystem where entities own and control their own identities[5]. Distributed systems and more specifically blockchain technology can help us to build trust in a system to make it possible. In the identity context, it is used to distribute an identity registry to every node in the network. The registry is only updated when nodes reach a consensus on the updated version. This way, not a single entity is able to modify it for its own interest.

1.1 Problem statement

How to trust people that you never seen and that you have to interact with online is a problem that we have been trying to solve for many years now. Trust in a digital ecosystem is a complex topic and a vital one to have healthy interactions between all its participants.

Trust is often linked to issues like security, respect of user privacy, technical reliability or authenticity of the information obtained in the internet. There is no standard way to know if an interlocutor is a human one or the one he is pretending to be. We need a solution that allows every Internet user to trust each other while maintaining their anonymity and their privacy. Can the self-sovereign identity model using distributed ledger technology provide a solution to this problem? That is the first question that we ask ourselves.

RQ1: Can self-sovereign identity improve trust in digital interactions?

When we want to build a persistent identity for users, the storage of this identity is very important. There are two important things we need to store. The cryptographic keys that are needed to authenticate the user as the identity owner and the credentials that will be associated to the digital identity. The more credentials you have, the more your digital identity will be rich. Moreover the recent GDPR legislation is giving us strict guidelines to follow about the management of personal information[6]. One of these guidelines is the ”right to be forgotten”, meaning that a user should be able to delete information that concerns him if he wants to. Because a blockchain ledger is by definition immutable, we need to store user's personal information in an off-chain separate component called ”agent”. Having a separate component to store private keys and credentials is an important concern for the future of this technology and we need to establish the best practices to implement it correctly.

RQ2: How can self-sovereign identity owners manage their identity with an agent, how do they access it and are they safe to use?

Today, when a person want to access a place or a service, this person often need to show one or more credentials. It is the case when taking a plane, you have to show a valid ticket as well as an identity document to prove your identity. While the identity document is used for authentication purposes, the ticket is used for an authorization purpose; you paid so you can access the service. It can work the same way with credentials associated to our self-sovereign identity. Because they are tightly linked to their identity, it is possible to verify that the user is the righteous owner of the credentials he is presenting. That is we name them ”verifiable” credentials. With our RQ3, we want to discover the mechanisms that allow a user to access a service using his verifiable credentials.

RQ3: Can verifiable credentials be used to manage users access rights?

The initial research question RQ1 was proposed by OBS (Orange Business Services) in order to discover the potential benefits of the blockchain technology for digital identity. We start exploring subject and got interested in the self-sovereign identity model. Even though I vaguely heard about self-sovereign identity in the blockchain engineering course held at TU Delft, it was not related to my project at that time and I left with very few knowledge of this technology. To discover if self-sovereign identity can improve trust in digital interactions but also to be ready to launch new services in case of an eventual shift towards this new model, we decided to make an overview of the underlying concepts and technologies and also looked at the efforts made to accelerate the adoption of this model, especially by looking at emerging standards and ecosystems.

As trust in digital identity being is highly related to the security of the infrastructure and strong authentication mechanisms, we decided to focus next on the use of agent in self-sovereign identity. Agents are essential components connecting user, blockchain and off-chain storage together and we want to discover how they work and if they are safe to use with RQ2. Finally, we ask ourselves in RQ3, if it is possible to manage users access rights using the credentials stored in their agents.

1.2 Research approach

As self-sovereign identities topic is fairly recent, the first step to answer these questions was to make an extensive study of self-sovereign identity and the associated verifiable credentials. By summarizing all the information we got, we provide a review of the technology in Chapter 2 to get a better understanding of the topic and start to answer RQ1 from a technical design point of view.

To address RQ2, we design and implement an agent in Chapter 3 that is a software component central to self-sovereign identities within an architecture where users are able to create an identity wallet where they can store their self-sovereign identity and digital credentials safely. We discuss authentication, storage, security as well as decentralization and recovery methods in case of agent loss. This allow us to determine the best practices to design an agent responsible of establishing secure communications with other identity owners and of the storage of our digital identities, credentials and cryptographic keys in digital wallets. We base our design on the Hyperledger Indy blockchain-for-identity technology[7] because it is the open-source solution where we found the most documentations but we hope our design can be applied to other solutions as well.

Finally, to answer RQ3 that is related to access management, we use the previous architecture and integrate a plugin to the agent in Chapter 4. This way, we hope to find out if it is possible to manage access based on verifiable credentials.

1.3 Thesis outline and Contributions

This thesis aims first to provide a global understanding of self-sovereign identity and verifiable credentials. In Chapter 2, we present both the background of digital identity and recent development as well as standardization efforts in self-sovereign identity. Chapter 3 is dedicated to the design and the implementation of an agent component allowing end-users to interact with the Hyperledger Indy blockchain. In this chapter, we also discuss authentication and the secure storage of cryptographic keys to answer RQ2. In Chapter 4, we introduce the idea of using digital credentials to create policies for access management and implement an access management component to answer RQ3. Finally, in Chapter 5 we present our conclusions and opportunities for future work.

With this work we hope to contribute to the development of self-sovereign identities, which we believe, would greatly benefit from both individual and organizations as well as enhancing the global user digital experience.

Chapter 2

Concepts of self-sovereign identity

The big data revolution was so fast that most of the people were unable to understand the value of their personal data in time. Most of the internet services today are provided for free in exchange of our data and we have no clue on how it is used but we're willing to give it away because these services are convenient. At the same time, central authorities are in charge of delivering certificates to decide who can be trusted or not. These central authorities represent central points of failure where someone with an access to it can corrupt the whole Internet. As awareness of these problems increased, a model where digital identity is centered on the user that is designed to give back the control on personal data and to emancipate users from central authorities is emerging. While there is no exact definition of what self-sovereignty is, we can use this terminology to describe individual or organizations being the sole owners of their digital identity. With the ownership of the digital identity comes also the control over how the associated personal data is shared and used. In this model, it is possible to collect claims about your identity from other peers or organizations to consolidate your digital identity. The authenticity of those claims can be verified and they can be used together as verifiable credentials to prove your identity to any other identity owner. By doing so, it is possible to trust your interlocutor without having to go through an intermediary. We can already find many ongoing project trying to make self-sovereign identity principles a reality. It is a will shared by corporations, universities and individuals alike and we are seeing interesting cooperation emerge to make it emerge as fast as possible. However, there is still technical and non-technical barriers to overcome before to be widely adopted. On the technical design part, wether or not this solution is able to improve our everyday digital interaction is what we are trying to answer in our first research question (RQ1).

As a background, in Section 2.1 we explain the evolution of digital identity to understand how the current identity model is working as well as to highlight that digital identity is complex and didn't stop to evolve. Then in Section 2.2, we discover the ten principles of self-sovereign identity which act as a guideline to build this new model. In Section 2.3 we present the architecture of self-sovereign identity as well as few related specifications before to introduce verifiable credentials in Section 2.4. In Section 2.5, we present few existing solutions already working on the implementation of self-sovereign identity. Finally, we analyze the macroeconomic factors of self-sovereign identity to know if the actual environment is favorable to the adoption of such a technology and to discover the barriers it has still to overcome in Section 2.6.

2.1 Evolution of digital identity

Digital identity has been and still is one of the core challenges of technological innovation. This challenge is all about enabling easier interactions between individuals and organizations in a way that is secure, private and more efficient[8]. Various efforts involving many institutions and individuals aimed to solve this problem. For each case, the goal was to represent existence within a certain context for trust and accountability to be possible.

Digital identity before the Internet

The first notion of digital identity appeared long before the Internet revolution. Before computers could communicate with each other, we already had massive databases owned and operated by governments, companies and banks. Their purpose was to model reality in a digital form to better manage and improve the process of accessing citizens, employees or customers data. To guarantee the uniqueness of a person, those institutions resorted to unique identifiers such as the national identification number[9]. As digital identity evolved, the fundamental mechanisms that were developed for those early systems remain the same today.

Protocol that allowed the Internet to scale

Then came the first version of what we call Internet today: ARPANET. Established in 1969, ARPANET was a network composed of universities and government computers[10]. At the beginning, addresses of physical hosts were managed via a single texfile ”hosts.txt” and maintained by the Stanford Research Institute. This solution was working well with the NCP (Network Control Protocol) designed for host-to-host communication within the same network and with few hosts. But in 1982, when the TCP/IP protocol which allowedw ARPANET to connect multiple networks, the population of hosts in the network exploded and eventually, it became impossible to maintain the hosts.txt file[11]. In order to solve this problem, the DNS (Domain Name System) was proposed one year later in 1983. The DNS is a distributed system maintained by the ICANN (Internet Corporation of Assigned Names and Numbers) and is serving the purpose of translating human-meaningful names into IP addresses[12].

Public key infrastructures

Eventually, as the size of the network grew, more security was required. To solve this problem, we used another important innovation of that time. The public key cryptography was discovered in the 1970s. Without it, we would not be able to secure public networks on which today's global communication and commerce rely[13]. Public key cryptography is using a linked pair of key. One of those key is public and the other is private. As their names indicates, the public key can be shared to anyone but the private key has to be kept secret by its holder. The private key cannot be deduced from the public key. It is allowing anyone to use the public key to encrypt messages that only the holder of the corresponding private key will be able to decrypt. However, that is not enough. In order to know which public key to use to encrypt your message, you need to know the name of its owner. This resulted in the creation of PKIs (Public Key Infrastructures) which issue and store digital certificates that attest that a public key belongs to a particular entity. Those PKIs are maintained by trusted third-parties called certificate authorities (CA) who are responsible of maintaining the integrity of the link between public key and entity[14]. Later, the PKI/CA technology was added to the web to create secure HTTP (HTTPS) enabling secure information sharing over encrypted channels[15]. It made the web secure enough to allow credit card transactions between organizations and individuals and allowed online commerce to grow significantly.

Social networks

At this point, interacting and transacting online essentially involved registering to websites giving them our data to access their services. Interestingly, the next major shift in identity is precipitated by social networks. The first factor that is precipitating this shift is a greater awareness of people regarding their digital identity, their personal data being directly visible to them. To the point that for many, online presence became a cultivated extension of the self[16]. The second factor is that social networks such as Google, Facebook, or LinkedIn began to take the role of identity providers by allowing people to connect to other websites using their social network identity. A convenient solution as you no longer require to register to every website you wish to access to. Instead, the social network is directly providing the user data to the website[17].

Problem with traditional models

There are few problems with the models discussed until now. The first one is the centralization of trust. We are unconsciously trusting third-parties when interacting with organizations online to whom we give our personal data that can be sensitive. These third-parties could be compromised[18] or lack integrity. A second problem is the fragmentation of our digital identity. When we register to different websites, we give them our personal data that are part of our digital identity[19]. The security of websites is not equal and it only take one breach in a less secure website for hackers to have access to our identity. Moreover, it is not user-friendly to fill registration forms with the same exact data each time we want to register to a website. Social networks through the identity provider model solve partly this problem but again trust and security can be an issue as you have to rely on a third-party.

Self-sovereign identity: an emerging alternative

In the next sections, we will discuss an alternative to traditional models called self-sovereign identity. It is a decentralized trust model based on blockchain technology which is a peer-to-peer network and is inspired from early works on PGP's Web-of-Trust[20]. We will discover the model's principles, its architecture as well as emerging solutions and the barriers to adoption.

2.2 Self-sovereign Identity principles

In order for to decentralized, a new model for digital identity is needed. In this new model, the user have to be at the center of the system and to be the own ruler of his identity. From this idea, came the term self-sovereign. To explain self-sovereign identity we will first look at its principles. In one of his essay, Christopher Allen introduces a list of ten principles to help us understanding what self-sovereign identity is about[21].

1. **Existence:** *Users must have an independent existence.*
2. **Control:** *Users must control their identities.*
3. **Access:** *Users must have access to their own data.*
4. **Transparency:** *Systems and algorithms must be transparent.*
5. **Persistence:** *Identities must be long-lived.*
6. **Portability:** *Information and services about identity must be transportable.*
7. **Interoperability:** *Identities should be as widely usable as possible.*
8. **Consent:** *People must freely agree to how their identity information will be used.*
9. **Minimalization:** *Disclosure of claims about an identity must be as few as possible.*
10. **Protection:** *The rights of individual people must be protected against the powerful.*

This list of principles is also a guide for organizations and individuals in an attempt of giving back the control over digital identity and personal data to their owner. In order to do so, this new identity model has to be user centered. The user-experience need to be improved and designed to empower him. However, digital identity always have been a complex subject and is difficult to manage. It is a double-edged sword that can be used for the best or for the worse. That is why the system must be carefully balanced taking into account all these principles.

As ambitious as it may seems, we are at the door of a shift in digital identity management and the self-sovereign identity model is actually the best contestant in updating the current model. In the next section, we will present the state-of-the art technology that makes this new model possible.

2.3 Self-sovereign identity's architecture

For many years now, proving our identity online as we do it offline has been a problem. Internet is an environment of distrust and the only available solution to add some trust has been to rely on mechanisms such as public key infrastructure (PKI). Few years ago, we had no ways to create an open PKI and we had to rely on centralized authorities to keep them secure. However, this centralized authorities represent central points of failure. Anyone able to access it could compromise the integrity of the entire internet[22]. The blockchain technology used in self-sovereign identity's architecture enables us to create the secure and open PKI that we need to redesign the entire concept of digital identity[23]. The applications it can provide are multiple. We have the possibility with this technology to build the identity layer which is missing to the internet and provide an universal identity for its users. An identity owned by the entity it represents which is secure and respect its privacy. And last but not least, an identity model which allow the collection and exchange of verifiable credentials with the possibility to verify digital signatures of their issuers.

In this Section, we take a lot of inspiration from emerging standard of the World Wide Web Consortium (W3C) because we think that standardization is a key to mass adoption of the technology. We will first present how a decentralized public key infrastructure works and what is the role of blockchain in it. Then, we will present the decentralized identifiers as well as other important specifications.

2.3.1 Decentralized Public Key Infrastructure

The first step, in order to return control of digital entities to their owners is to find an alternative to traditional PKIs which have many usability and security issues. Today, a big part of the web traffic is unsigned and unencrypted because of its complexities compromising user privacy and burdening organisations with huge liability risks[24].

One of the alternative is the Decentralized Public Key Infrastructure (DPKI). We can find the mechanisms of the DPKI in the paper of the same name cowrited by experts in digital identity and decentralized systems[22]. Unlike PKI, the goal of DPKI is to ensure that no single third-party can compromise the integrity and security of the system. To achieve this goal, DPKI is using blockchain technology that is a decentralized key-value store and make possible geographically and politically diverse entities to reach consensus on the state of a decentralized ledger (Figure 2.1). We often think that blockchain is a way to remove the need for third-parties. It is not true, we still need third-parties but they do not endorse the same roles. In blockchain ecosystems they are called miners[25] and their function is to ensure the security and integrity of the ledger. They are financially incentivized to do so and can be financially punished if they do not respect the rules of the protocol.

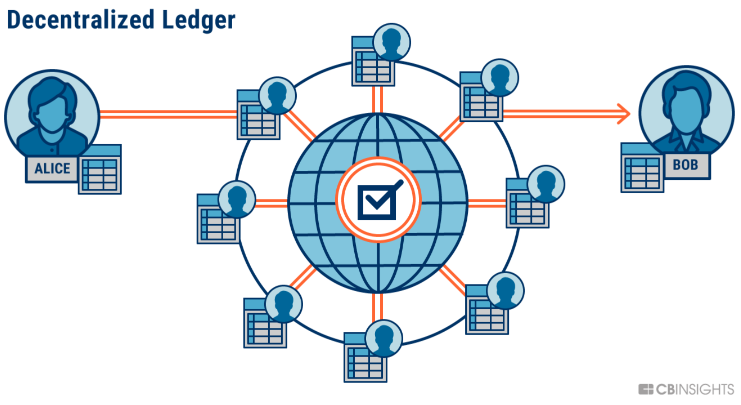


Figure 2.1: Representation of a decentralized ledger

This decentralized ledger allows the registration of identifiers that can act as look-up keys for arbitrary data such as public keys. Control of identifiers is returned to the ones who made the registration and it becomes no longer possible to compromise an identifier that is not yours. In contrast to tradition PKIs, where an attacker needs to compromise one CA to affect many identity owners. The attacker would have to attack each identity owners he wishes to compromise within the DKPI. The only other way to compromise many identity owners at a time would be to compromise the blockchain itself which is extremely difficult as the attacker would need to have control over 51% of the network[26].

In the next subsection, we will have a closer look at the decentralized identifier mechanisms.

2.3.2 Decentralized identifiers

To represent digital identities in a decentralized environment, a new kind of identifiers have been thought of to fit this model. Those decentralized identifiers (DIDs) inherit interesting properties from the blockchain. They are independent for any centralized registries (decentralized), persistent and is it possible to prove their ownership using public-key cryptography[27].

DID Format

DID format is similar to Uniform Resource Name (URN) format as we can see in Figure 2.2[28]. The key difference is that we are specifying a method instead of a namespace in the DID format. The method is referring to the specification used to register and manage DIDs on a particular blockchain or DLT (Distributed Ledger Technology). From this point, we will only use the term DLT which is a more generic term that includes blockchains. The DID method has to be conform with the general DID specification and must define the format and generation of the method-specific identifier that have to be unique within the method space. The DID can be resolved to a standard resource named DID Document that describes the entity it represents.

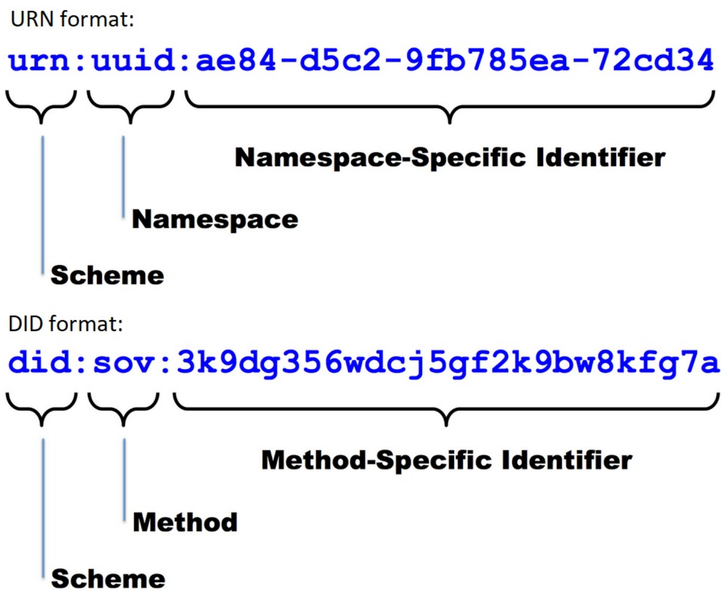

URN format:
urn:uuid:ae84-d5c2-9fb785ea-72cd34
Namespace-Specific Identifier
Namespace
Scheme
DID format:
did:sov:3k9dg356wdcj5gf2k9bw8kfg7a
Method-Specific Identifier
Method
Scheme

Figure 2.2: DID format (Sovrin method) compared to URN format

DID Documents

We can see DID infrastructure as a global key-value database where the database consists of all DID-compatible DLTs. In this global and distributed database, the key is a DID, and the value is a DID document. A DID document is a JSON-LD file which stand for JavaScript Object Notation for Linked Data. The main advantage of JSON-LD is interoperability[29]. Adding an ”@id” and a ”@context” to the original JSON format make possible for other documents to be linked with the

DID owner and remove any ambiguity concerning the document's fields because they have to be described by the context. In the DID context, the JSON-LD object includes six core components that are necessary to interact with the owner of the DID specified in the DID Document:

1. The DID itself, so the DID document is fully self-describing and we can link other documents to this DID.

2. A set of public keys or other proofs that can be used for authentication or interaction with the identified entity.

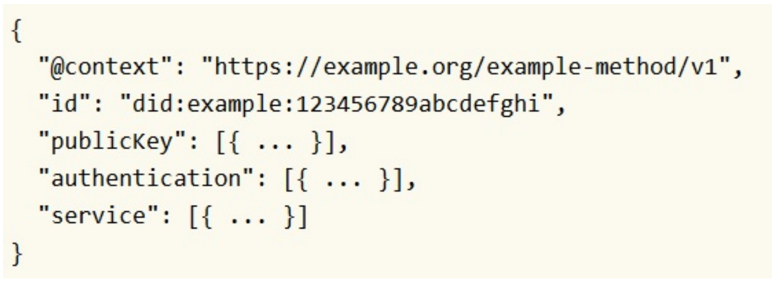
3. A set of service endpoints that describe where and how to interact with the identified entity. A service endpoint could for example link to the mail service of the DID's owner or to his storage service.

4. The owner of the DID can also decide to authorize other entities to make changes to the DID Document. This can become particularly important and useful in the case of private key loss.

5. Timestamps can be added for auditing the date of creation or update of a particular DID.

6. A JSON-LD signature can also be added if there is a need to verify the integrity of the document.

In Figure 2.3, we have an overview of a simple DID Document example. To find a more complete it is possible to go on the World Wide Web Consortium's DID specification page directly. W3C is an international organization in charge of developing internet standards and it is important for Orange Business service to follow standardization efforts. That is why we often referred to the W3C standard to explore self-sovereign identity.



```
{
  "@context": "https://example.org/example-method/v1",
  "id": "did:example:123456789abcdefghi",
  "publicKey": [{ ... }],
  "authentication": [{ ... }],
  "service": [{ ... }]
}
```

Figure 2.3: A simple DID Document example from the W3C's DID specification.

DID Methods

As mentioned earlier, the DID format contain a field from which we can determine the method used for its creation. DID methods purpose are transparency and interoperability. They must be specified for other entities to understand how the DID and DID document are created, resolved, and managed on a specific DLT so they can evaluate and include your method in their project. To be accepted, a DID method specification must contain the following elements:

1. The name of the method, a short description and the targeted DLT.

2. The ABNF structure (a standard for formal syntax) of the method-specific identifier and the way it is generated.

3. The JSON-LD context used in the DID documents

4. The way operations Create, Read, Update and Delete (CRUD) are performed on a DID or DID Document. The operations must be detailed enough for a client to be able to test and build its implementation on the targeted system.

5. The security and privacy considerations to take into account when using this specific method.

The CRUD operations are the four basic functions of persistent storage[30] and can be used to perform all the operations required by traditional CKMS (Cryptogtaphic Key Management Systems) such as key registration, key replacement, key rotation, key recovery and key expiration[27]. The CRUD operations is the part that may vary the most between the different DID method specifications due to the specific nature of the chosen DLT. You can find the list of the CRUD operations below:

1. **Create:** The method must specify how to create a DID and the associated DID Document on the targeted DLT as well as the way to prove its ownership.

2. **Read/Verify:** The method must specify how to use a DID to request the associated DID document and verify the authenticity of the response.

3. **Update:** The method must specify how to update a DID Document including the way to prove your authorized to do so.

4. **Delete/Revoke:** The method must specify how to revoke a DID including the way to prove its revocation. It is more adapted to speak of revocation since DLTs are by nature immutable. It is done by updating the DID Document to null.

2.3.3 Associated protocols specifications

The decentralized identifiers (DID) specification is one of the specification that will help us to standardize decentralized identity architectures. Other protocols and specifications are currently developed in an effort to standardize decentralized model of digital identity in order to provide interoperability between the different solutions.

Decentralized Key Management System

Decentralized Key Management System (DKMS) is an emerging open standard for managing users DIDs and private keys. The idea is to standardize digital agents and wallets in order to facilitate the user experience. Inside the DKMS specification, we can find elements that we discussed before such as DIDs and the linked DID Document as well as the verifiable credentials that contains all the necessary information to interact with external entities. There are multiple benefits to adopt the DKMS specification in self-sovereign identity systems. In this specification, we use a cloud layer with cloud agents as a medium between the user, his edge agent and the DID Layer (or DPKI) as we can see in Figure 2.4. This architecture makes easy for users to have multiple wallets across different devices and also provide identity owners with an agent that will be available at any time to receive their communications. Another benefit of the cloud layer, is to store a backup of the user's digital identity and provide recovery in case of private key loss (see Section 3.5.2). Moreover, having a standard for agents and wallet would improve greatly the control of he user and the portability of his data, as he could switch from one provider to another more easily.

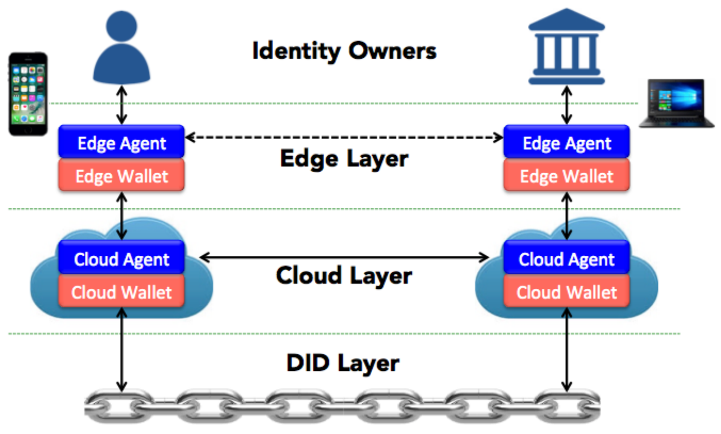


Figure 2.4: Decentalized Key Management Systems - Rebooting the web of trust

DID Transport Layer Security

The DID Transport Layer Security (TLS) protocol specification is a decentralized version of the current TLS protocol which uses X.509 certs and is based on traditional PKIs. While the current TLS protocol is securing the communication between a client and a server, the objective of this protocol is to provide encrypted peer-to-peer connections in real time between two DID owners (whether they are individuals, things or organizations). It could greatly improve the security provided by the current TLS protocol in our online interactions since it could be applied to all form of online communication. To make it possible, we need to enable the dynamic creation of x.509 elements from any DID and DID document to establish a dedicated TLS session.

DID Auth

The goal of self-sovereign identity systems is also to allow the cryptographic authentication of a DID owner. The DID Auth protocol is inspired by the Secure Quick Reliable Login (SQRL) open standard and the Web Authentication protocol which both use a cryptographic challenge to authenticate the user. In these protocols, the challenge is sent by a relying party to a user who sign it with its private key. Then, the relying party can check if the signed challenge is corresponding to the user public key. While the ownership of the public key cannot be verified in these two protocols, it becomes possible for relying parties to verify it by resolving DIDs and checking the DPKI used to create it. The standardization of this specification would allow the authentication of all identity owners using a DID that supports the specification and promote interoperability.

DID Names

We have seen above that the DID format is not something we can easily remember. It is not ”human-meaningful”. In fact, a trilemma known as the Zooko’s triangle tells us that an identifier cannot be human-meaningful, decentralized and secure at the same time. An identifier can only have two of those properties. For in order for an identifier to be secure and decentralized it can’t be human-meaningful[31]. However there are many use cases where it is desirable to be able to discover a DID using a human-meaningful identifier. You would have to give up either the security or the decentralization but that is not what we want.

To solve this problem, like for domain name in the past, a DID naming system is needed. But this time it should be entirely decentralized and not be based on centralised registries. The goal of this protocol’s specification is to standardize to how operate a DID naming layer on top of the DID layer. In order to achieve this, DID names should be registered in the blockchain and linked to the corresponding DID the same way DID Document are linked to DIDs.

2.4 Verifiable credentials

Verifiable credentials are another concept that make self-sovereign identity desirable as a new model for our digital identity. Credentials often contains information on its holder (e.g. name, age, address, etc.) as well as information on the entity who issued it. For some of them an expiration date is also needed. If we think about it, credentials are everywhere. We get our first one, the birth certificate, very early and we are collecting more of them all along our life. However, they are mostly issued as piece of paper or plastic card and we have no standard way to assert their authenticity online. Or to be more accurate, no good way to do it. The verifiable credentials w3c's specification provides us guidelines to use credentials online in a way that is cryptographically-secure, privacy-preserving and machine-verifiable. Even though, self-sovereign identity is not necessary to implement verifiable credentials, we will explain how they are complementary.

In Subsection 2.4.1, we present the roles that need to be assumed by actors to create a verifiable credential environment. In Subsection 2.4.2, we present the different type of information that we can find inside a verifiable credential and how they're presented in Subsection 2.4.3. To finish with verifiable credentials, we explain the mechanisms that allow them to be verified in Subsection 2.4.4.

2.4.1 Verifiable credentials environment

For verifiable credentials to be useful, we need first to create an environment where different actors can interact with each other using these credentials. We can dissociate four roles that actors can assume to create this environment as we can see in Figure 2.5.

- **Holder:** The holder's role is performed by acquiring verifiable credentials from issuers and storing them in order to later present them to relying parties who request it.
- **Issuer:** The issuer's role is performed by creating verifiable credentials and associating them with a specific entity (i.e a holder).
- **Verifier:** The verifier's role is performed by requesting verifiable credentials from an entity. It allows the verifier to ensure that the holder is fulfill some requirements.
- **Identifier Registry:** This role is different from the three others as it is performed by a computer system. In the first place, this system allows the holder to register an identifier that will allow him to interact with issuers and verifiers. Then, these latter will be able to use this registry to verify that the holder's identifier is its own before to issue or to request a verifiable credential.

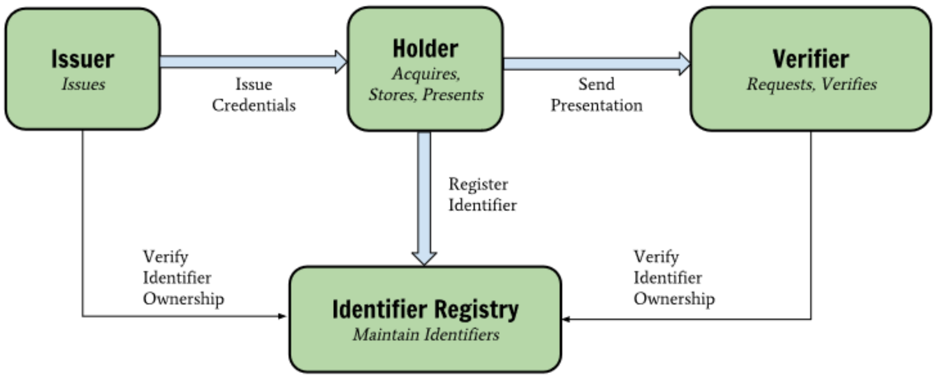


Figure 2.5: Verifiable credentials environment from the W3C Verifiable Credential data model specification.

That is on this last role that the self-sovereign identity model can be of use in this ecosystem. Altough this role can be fulfilled by other systems such as government's or other trusted databases, the self-sovereign identity model with its DPKI that use distributed ledger technologies to register identifiers (i.e. DIDs) is the best candidate to assume this role.

Is is the best candidate for few reasons. First the trust is decentralized, the identifier are controlled by their owners and does not depend on any centralized authority. Decentralized identifiers are also persistent which mean that as long as you have the associated private key you will be able to authenticate as the credential's holder in a way that is cryptographically-secure (c.f. DID Auth). Finally, DPKIs are public and transparent which facilitate the onboarding and self-management of its participants. More actors there is in the ecosystem, more the verifiable credentials have value as the interoperability increases and users could progressively be able to use them for all their online interactions.

In Table 2.1. you can find examples of credentials and the associated actors and the roles they are performing for this specific credential.

Credential's type	**Issuer**	**Holder**	**Verifier**
Degree	University	Student	Employer
Driving's licence	Government	Driver	Police
Medical records	Doctor	Patient	Other doctors or pharmacy
Reusable KYC*	User (self-issued)	User	All websites that require it
Company's status	Government	Company	Tax authorities

Table 2.1: Credential's type and the associated entities' roles

*The reusable KYC (Know Your Customer) credential is a bit special because the user is the same time issuer and holder. For websites that do not require high-level security, the user can make claims about himself for the field we usually manually fill when registering to a website (e.g. name, surname, email, mail, etc.) and use this credential each time he wants to register to a new website. It is a simple example of how verifiable credential would enhance user experience in our daily online interactions.

2.4.2 Data model

In order to create trusted and scalable digital credentials, authors of the specification designed a model flexible enough to fit any use cases that require credentials in the physical world and ensure that this model would allow actors of the ecosystem to interact safely with each other.

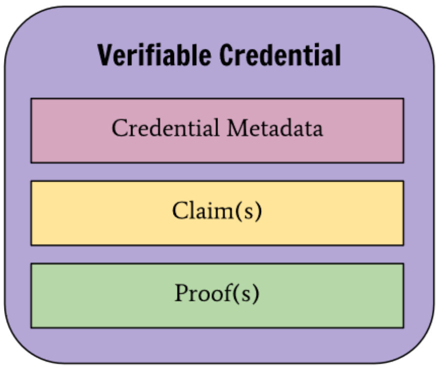


Figure 2.6: Structure of a verifiable credential from the W3C Verifiable Credential data model specification.

In figure 2.6, we can see the model they design for verifiable credential is composed of three main components :

- Credential Metadata used to describe the property of the credential such as the identifier of the credential, its type (e.g. Degree Credential) and its issuer (e.g. University). We can add other type of information like the issuance and/or the expiration date to adapt to different use cases.
- A claim or a set of claims about the subject. Claims are statement about a subject and can be expressed using subject-property-value relationships. When an university is issuing a degree to a student, it is equal to making the statement that a student (i.e. the subject) has graduated from (i.e.the property) the university (i.e. the value).
- A proof or a set of proofs used to verify the authenticity of the credential. Proofs include a type, a reference to the signing entity, the signature value

and the date of the signature.

The verifiable credential can also be visually represented as a combination of two graphs as we can see in Figure 2.7. The first graph is representing the credential itself and include metadata and claim(s) about the subject. The second one is representing the digital signature of the credential. We can verify multiple elements of this signature such as the type of signature, who created it and when as well as its value to prove its authenticity.

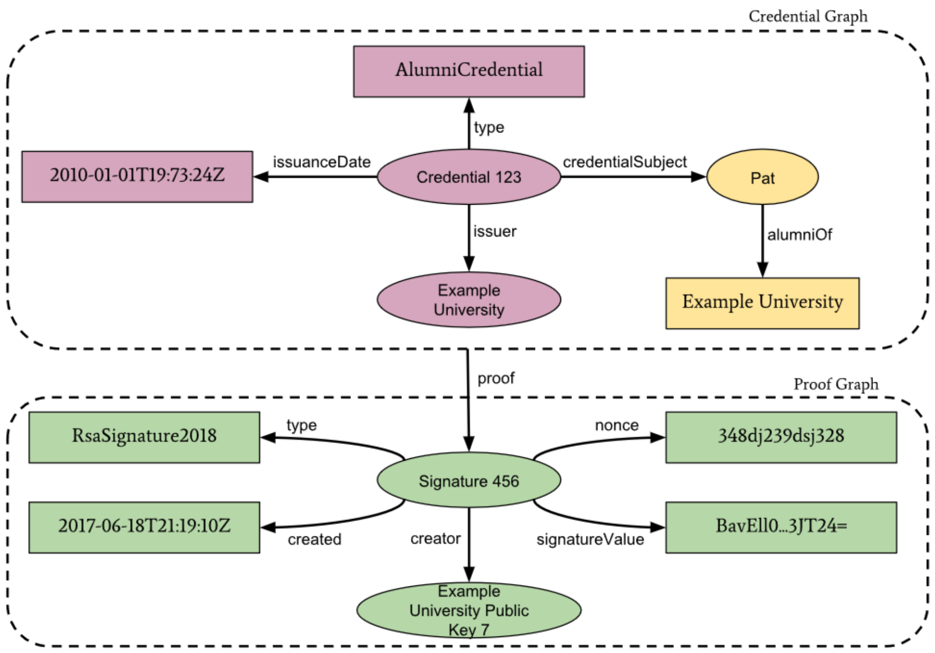


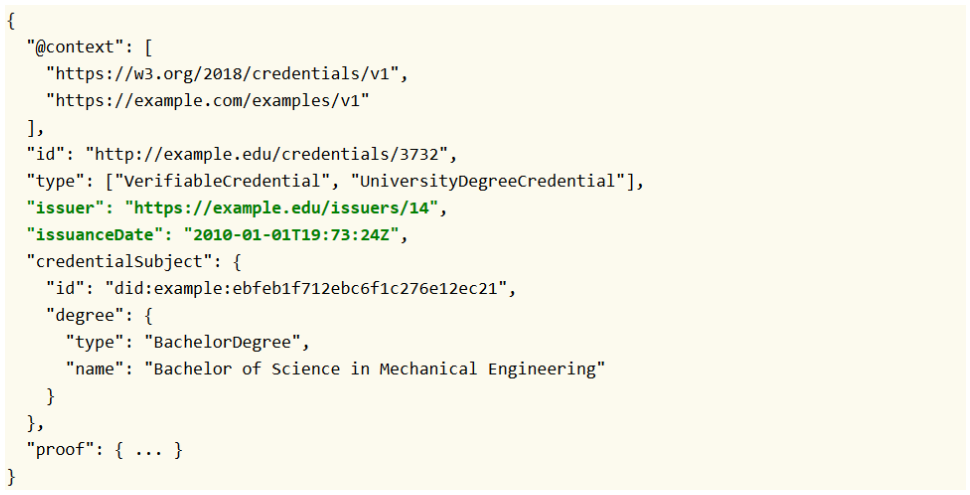
Figure 2.7: Verifiable credential graph from the W3C ”Verifiable Credential Data Model 1.0” specification.

2.4.3 Format

As for DID Documents, verifiable credentials are expressed in a JSON-LD file from which we can have a preview in Figure 2.8. In this file we can find all the components described by the data model. We can note that the subject is represented by his DID. This relation is the link between the identity owner of self-sovereign identity and verifiable credentials together.

2.4.4 Verification

There is a number of check that are necessary to verify a credential. Some of them are essential while others are optional depending on the type of credential we are verifying.



```
{
  "@context": [
    "https://w3.org/2018/credentials/v1",
    "https://example.com/examples/v1"
  ],
  "id": "http://example.edu/credentials/3732",
  "type": ["VerifiableCredential", "UniversityDegreeCredential"],
  "issuer": "https://example.edu/issuers/14",
  "issuanceDate": "2010-01-01T19:73:24Z",
  "credentialSubject": {
    "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
    "degree": {
      "type": "BachelorDegree",
      "name": "Bachelor of Science in Mechanical Engineering"
    }
  },
  "proof": { ... }
}
```

Figure 2.8: Verifiable credential JSON-LD file from the W3C ”Verifiable Credential Data Model 1.0” specification.

1. **Syntax:** The document must be a syntactically valid JSON-LD file.
2. **Credential:**Properties such as credential’s type and proof that are required in the w3c specification must be present.
3. **Issuer:** The verifier must be able to verify that the issuer is known and trusted. He can use the issuer metadata such as is public key to do so.
4. **Subject:** The subject or holder of the credential must be identified with his did. Public metadata related to this DID should allow the verifier to authenticate the subject using its signature.
5. **Proofs and signatures:** To prove that information in a verifiable credential is correct, we use cryptographic mechanisms called proof. For different type of proofs such as zero-knowledge-proofs, proof of work, proof of stake or digital signatures; the verifier must make sure that the proof is a known proof suite and all properties required by this suite are present. If the proof is a digital signature, the verifier must make sure that the public key associated is valid and owned by the right entity.
6. **Issuance:** The verifier must verify that the issuance of the credential was performed before the verification date.
7. **Expiration:** The verifier must verify that the credential has not expired if the credential contains an expiration date.
8. **Revocation:** If some revocations instructions are present on the document, the verifier must make sure that the credential have not been revoked.

9. **Custom:** For specific use cases, the verifier should be able to make define custom verification.

These verification processes are essential in order to guarantee trust between the actors within the decentralized ecosystem.

2.5 Existing self-sovereign identity solutions

We can see today that there is a race to develop functional decentralized identity system. In the website of the Decentralized Identity Foundation (DIF) we can see that over fifty companies joined the foundation in the year of its creation. The purpose of the DIF is for its members to work together on different aspects of decentralized identity such as decentralized identifiers; discovery, storage and computation of the attributes or the implementation of verifiable claim and credentials.

In this section, we give an overview of three among many solutions that were studied to design our credential repository. We selected these Sovrin, uPort and TrustChain solutions because they best matched the self-sovereign identity principles[32], were open-source with an acceptable amount of documentation.

2.5.1 Sovrin (Hyperledger Indy)

Sovrin is a distributed ledger which was specially built for decentralized identity. This project was initially developed in 2015 by a startup called Evernym in order to solve the online identity problem and to enable the exchange verifiable claims they built the blockchain called Sovrin.[33]

In a concern of universality, they decided afterwards to donate their codebase to a non-profit foundation which is known as the Sovrin Foundation. The code being released as open-source they continue to upgrade the technology with help from the community. Later, the codebase was once again transferred from the Sovrin Foundation to the Linux Foundation in 2017 to become the Hyperledger Indy project[7].

2.5.2 uPort (Ethereum)

The next self-sovereign identity solution we discuss about is called uPort. It is based on the Ethereum blockchain and allow users to register their own identity on the ethereum blockchain using smart contracts. It also allows identity owners to send and request credentials, sign transactions, and securely manage keys & data via their mobile application[34].

The code of the project is open source and it use both did and verifiable credentials standards inspired from the w3c's specification. Due to the nature of Ethereum the scalability is quite low and transactions can take time to be accepted.

2.5.3 TrustChain (IPv8 protocol)

TrustChain is a promising technology built in the Blockchain-Lab of our university TU Delft. TrustChain is a complete solution which allow the user to create and store your digital identity in your smartphone[35]. In 2018 with the cooperation of the Dutch government, experiments were conducted at small-scale in few selected municipalities where citizens could request the first of kind self-sovereign identity document[36].

Unlike the two other solutions, TrustChain's architecture make use of multiple chains (one per identity owner) to build their system. They also decided to not follow the w3c standard because, from their point of view, the metadata used in the standard's credential can leak sensitive information about the identity owner.

2.6 Overcome the barriers to adoption

A large part of my internship was to identify the barriers self-sovereign identity still has to overcome in order to be adopted and to evaluate the chance for the technology to be used by businesses.

To get an answer to these questions, we first analyze the macro-environmental factors in Subsection 2.6.1 then we provide an evaluation of self-sovereign identity in Subsection 2.6.2 to conclude this chapter.

2.6.1 Analysis of the macro-environmental factors

Before to conclude on this survey, we analyze the macro-environmental factors that affects the adoption of self-sovereign identity. In order to make our analysis, we use the PESTLE framework that will help us to see the big picture. There are seven factors described in this framework: Political, Economic, Social, Technological, Legal and Environmental[37].

Political

Political factors are very likely to impact self-sovereign identity project. One of the reason is that until now citizens identity was provided by the governments and those projects could create an alternative. In self-sovereign identity, governance is decentralized and governments could not like this idea.

However, it is not necessarily opposed to traditional systems. It could even ease the work of the governments to secure and reduce fraud. Some countries like Netherlands[36] and Estonia[38] among others are pioneers in the field and are starting to experiment those technologies.

Economic

In early stage of development, the economic factors that impact the development of self-sovereign identity solutions are the amount of funding related projects can get for R&D, tests and experiments. Self-sovereign identity infrastructure is very likely to reduce the cost for Identity, Access Management and Cybersecurity in the future due to its digital and distributed nature. Therefore, the price of the solution should not affect its adoption by businesses negatively.

However, to make a transaction on the blockchain you often need to pay a small fee. In order for people to adopt these solutions more easily, a business model where users don't have to pay fees each time they're interacting with the system would be more appropriate.

Social

Recent development of the Internet led us to the big data era that is characterized by the production, collection of a massive amount of data that is processed to produced meaningful information. Data collected from people is a part of their digital identity and can become sensitive if not handled the right way. While it is not a concern for most of people, it is important to raise awareness on this question and self-sovereign identity can be a solution to secure and manage this data.

We can also notice a behavioral shift as people are using more and more the web and their smartphones to interact with organizations such as banks or insurance companies. Self-sovereign identity solutions can be used to facilitate those interactions thanks to its user-centered model. Using such a model with collection of information as verifiable credential in one place provides an easy onboarding and a secure access to services by facilitating the processes of information verification.

Moreover, by allowing people to control their identity but also to issue and verify credentials from other identity owners lay the foundation for more cooperation between peers.

Technological

The technology is still at an early stage. At the moment, most companies are trying to explore what we can do with it but we cannot see large adoption of these solutions yet. Working at the edge of technology have pros and cons. Even if you can find few open-source project on the topic (i.e self-sovereign identity), documentation can be hard to find or incomplete. Moreover, such a system require the interactions of multiple components and it can be difficult to make them work together. In the other side, managing to go through the complexity will give you a competitive advantage as it is not something that can be replicated easily.

In term of adoption, using the newest technologies allows to attract the attention of early adopters like businesses and tech-enthusiasts.

Legal

When dealing with users identity and personal data, there is many legal restrictions. At the international level, we have the GDPR (General Data Protection Regulation) that came into force in May 2018. The regulation is targeting ”any information relating to an identified or identifiable person” that lives within the EU [6]. It is aiming to enhance individual privacy and give more control over their data to European citizens. A second objective is to enable portability of personal data across the EU to create trust that should allow economy to develop.

The ”eIDAS” is another legislation that apply to EU countries. This time again, the objective is to increase trust in digital transactions[39]. The legislation defines rules about online identification and about the legal status of electronic document with the use of mechanisms like digital signature for example.

Self-sovereign identity’s principles are totally aligned with the objectives of those regulations. In that sense, the legal context is suitable to the development of such solutions.

Environmental

Today, identity and administrative documents are still, for the main part, issued in paper or plastic format. Digitalizing these processes would be beneficial for the environment as it would remove the need for these resources.

However, in order for a distributed network to be secure, consensus algorithms have to be run. Proof-of-Work (PoW) consensus used in Bitcoin, Ethereum and many other blockchains is not really environment-friendly due to the amount of computation power needed to run it and has a carbon footprint equivalent to the one of Las Vegas[40]. However, other types of consensus algorithm such as Proof-of-Stake are less power-consuming and could be used to reduce the impact on the environment.

2.6.2 Self-sovereign identity evaluation

Like in the past, digital identity will continue to evolve according to the evolution of the professional and/or personal needs of Internet users. Today, the guiding principles of self-sovereign identity principles is what users need and will empower them as well as improving their experience online.

From our analysis of macro-environmental factors, we can say that there is a favourable climate for self-sovereign identity model to be implemented. Especially on the social and legal aspects, with the society being more aware than ever on the existing privacy issues, the existing legislation on personal data protection and electronic documents to regulate it. Political factors are more country-dependent, we can see some country already experimenting the technology while other are experimenting different solutions to increase trust in digital documents. Economic and environmental factors are neutral, they have both positive and negative points. As for the technological factor, we have seen that the technology is still at an early stage but there is plenty of ongoing project trying to design the right solution as well as standardization effort creating a positive environment for collaboration and giving guidelines for developers interested in the subject. In any case, that is the social aspect that should be prioritized over the other aspects. If this solution can solve digital identity issues, the best is to find the right design and the right business plan to implement it in our society.

Chapter 3

Design and implementation of a self-sovereign identity agent

After concluding our survey on self-sovereign identity and verifiable credentials, our goal is to realize a prototype of a service where a user is able to create his identity and start to gather digital credentials to enrich it. There are two important things the user need to store that require a high level of security. The first of those two things are the cryptographic keys and especially the private key. It is because anyone in control of the private keys can perform actions in the name of the identity owner. These keys are also needed to perform all encryption and decryption actions and guarantee that the user information is communicated in a secure way. The second one is digital credentials that contain personal information and that can be used to access resources and services. In Chapter 2, we have learn about the agent component that is in charge of and described in the Decentralized Key Management System (DKMS) specification. In this chapter, we try to learn more about this component that is central to self-sovereign identity and answer our second research question (RQ2) on identity management and the safety of this component.

In Section 3.1, we set our design goals for our agent implementation. In Section 3.2, we give an overview of the architecture that we used to meet our design goals. Then in Section 3.3, we present the different type of agents that can be built and their mechanisms. In Section 3.4 we present the implementation of our own agent and we end this chapter by giving general remarks about agents on trust, security and decentralization in Section 3.5.

3.1 Design goals

We start by setting design goals for our prototype of a self-sovereign identity agent. These goals help us to design the correct architecture that will help us to answer our second research question (RQ2) concerning the use of an agent. Here is the list of our goals:

1. **Identity registration:** Users or identity owners must be able to register their own identity (a DID) via our web service and use it to request, issue and verify credentials. Afterwards, they must be able to manage their credentials via an interface.

2. **Authentication:** Identity owners must be able to authenticate as owner of their identity and access their identity wallet.

3. **Storage:** Wallets of identity owners can either be stored in off-chain or online agent. In our design we want to propose an hybrid solution to guarantee security, reliability and availability.

4. **Account recovery:** If the user loses his access, he should be able to recover his account.

5. **Credential issuance and verification:** To test our service, we must simulate the role of different actors. Some actors will act as credential issuers and verifiers.

In the next section, we present the architecture we have designed to reach thesedesign goals.

3.2 Architecture overview

We will build our prototype using Hyperledger Indy (see Section 2.5.1). We chose this technology for several reasons. The main one was the amount of documentation and tutorials available. Then, it has a large ecosystem and if the prototype is satisfying we can work directly with Evernym, the company who initiated the Hyperledger Indy project, to build our solution faster. Finally, a team of OBS was working on a project with Hyperledger Fabric and we thought it could help with the development of this prototype. In the Figure 3.1, we show an overview of an architecture which includes the following components:

1. **Identifier registry (DPKI):** The first step is to build the identifier registry that we created from a network of 4 indy validator nodes. Each of these nodes contains a distributed ledger that is kept in sync using a consensus algorithm. In Hyperledger Indy, the consensus algorithm is called Plenum and is based on the RBFT (Redundant Byzantine Fault Tolerance)[41] algorithm.

2. **User Agent:** An agent is a component central to self-sovereign identity. It is a software part, like a mobile or web application that provide an interface as well as a wallet and communication protocols including the Agent-to-Agent (A2A) protocol so users can start interacting with other identity owners. There can be only one agent per device. For our prototype, we will build four agents that each represent an actor of an educational environment.

3. **Identity wallet:** In the self-sovereign identity ecosystem, identity owners are storing all the information related to identity in a wallet. These wallets allows the secure storage of cryptographic keys and credentials. Hyperledger Indy is following the emerging DKMS open standard discussed in Chapter 2 that guides developers in the design of these wallets.

4. **Interface:** In order to interact with their wallet and the identifier registry, identity holders need an interface to abstract the complexity of the system.

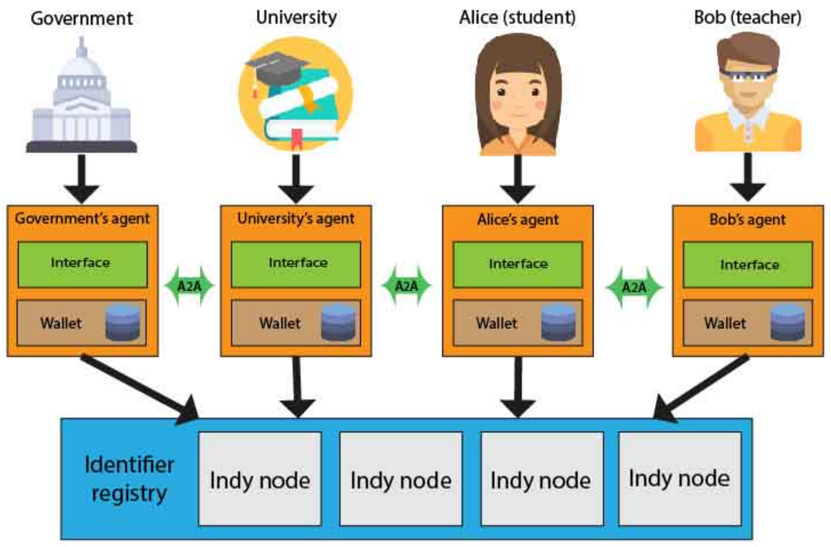


Figure 3.1: Self-sovereign identity service architecture

In the next Section, we will give more details about the different type of agents we can find and explain how to authenticate to an agent, where does it store your identity wallet and the different keys that are used to manage it.

3.3 Agents to host identity wallets

To start with self-sovereign identity and verifiable credentials, the user will need an identity wallet to store his cryptographic keys and his credentials. Because humans are not able to store data and to use cryptography on-demand, we need to delegate those tasks to an agent that will manage our identity wallet in our behalf. Agents can be used in many different ways but they all share the three following characteristics:

- They act on the behalf of an identity owner.
- They hold cryptographic keys to perform actions delegated to them by the identity owner.
- They can interact with each other using an agent-to-agent protocol.

An introduction to the agent concept can be found in the Indy Hype github repository [42] that aim to standardize processes within the Indy ecosystem. However, those concepts can inspire other ecosystem facing the same challenges.

We first describe the different categories of agent we can find in Subsection 3.3.1. Then in Subsection 3.3.2, we discover where the identity wallet is stored within the agent. In Subsection 3.3.3, we give details about the different was to authenticate to an agent. Finally in Subsection 3.3.4, we give details about the different key managed by the agent.

3.3.1 Agent categories

As stated earlier, an agent can be used in various ways, we can categorize them using different factors. This categorization will help us to decide what kind of agent is needed for our solution. We can also use this category in the future to provide personalized agent depending of the need of our customers.

Trust based

Categorizing agent by trust helps us to decide what we can delegate to an agent or not. An identity owner should be able to fully trust an agent that runs in an environment that is in his direct control. This agent can be considered **trustable**. If it runs in an environment that can be accessed by other people, then it is considered **semi-trustable** and you should think twice before to delegating critical authorization to this agent.

However, users environments are not necessarily more secure against hackers because service providers has more resources to secure their own environment. Still, a service provider can be more subject to attacks or a malicious system administrator could compromise your agent.

Location based

An agent can be host either on your device either on a cloud infrastructure. If it's hosted on your device, we will call it an **edge agent**. If it's hosted we will call it a **cloud agent**.

An edge agent can be referred as cold storage and is considered trustable. It is theoretically the most secure way to store your identity wallet because only someone with direct access to the device can access the wallet. There is few limitations with edge agents. If you are out of battery, if there is a technical problem affecting your device or if you lost it; you could lose temporarily or indefinitely your access to the wallet if you did not take any measures to back it up.

Cloud agents can be either trustable or semi-trustable depending if the cloud environment is in your control or is provided to you as a service. They can overcome the limitations of edge agents. The main advantages of cloud agents are availability and reliability, meaning your wallet will always be available from anywhere and with any device. It also possible to perform tasks like encryption, key management, data management, data storage and backup that are difficult to perform on an edge devices due to their technical limitation (e.g., computational power, bandwidth or storage capacity).

Complexity based

Depending on the capacity of your device, your need or your use case you may use agent with a different level of complexity.

- The **static agent** is the simplest agent and is configured for a single relationship. This agent is most of the time a bot programmed to execute a single task.
- The **thin agent** also focus on one relationship context that can be updated over time. This agent can be an app that is used by a customer to interact with his bank.
- The **thick agent** is able to handle multiple relationships but still focus on one specific use case. This agent can be a social app where you have to connect with multiple users or it can be used by the bank of the previous example to interact with multiple users.
- The **rich agent** is the more complex, it is able to manage all relationships and credentials in one place. It supports more functionalities than the other agent ans can be used in many use cases.

3.3.2 Agent storage

As we mentioned before, the wallet will be hosted by an agent and store our cryptographic keys and credentials. While most of the agent include wallets, it's not mandatory. Because some devices have a limited storage capacity, it is possible to delegate your wallet to a cloud agent to store your credentials. However, for most of the case you would have an encrypted wallet stored in the device where the agent software is installed. By default, the wallet is stored in an sqlite database when using the Hyperledger Indy SDK but it is also possible to set a custom storage configuration.

3.3.3 Authentication to agent

No matter the kind of agent the user chose, he needs a way to authenticate to this agent in order to perform actions. But depending on the type of agent you will need different authentication mechanisms. There is three authentication factors and in each case we have one or a combination of these factors:

- Something the user know (such as a password)
- Something the user have (such as a smart card)
- Something the user is (such as a fingerprint)

An additional factor of authentication could be location-based: Where the user is. But we'll focus on the three main factors of authentication.

Direct authentication to edge agent

As the edge agent is stored in your mobile or computer, only someone with direct access to this device will be able to access agent (something you have). Still if you lose your device or if someone has to access it for other reasons you still want to prevent them to be able to perform action with your agent. In order to secure the access, we can add either a password, a pin code or a pattern (something you know). This way, only you can access to the agent; at least, if you didn't disclose this information or that the password/PIN/pattern is not too easy. To be sure that only you can access the agent, you can eventually add a fingerprint or other biometric method (something you are).

In most of the devices, these passwords or biometric information will be hashed and salted before to be stored into your device's filesystem. This way, it is very difficult for a hacker to get access to your agent because he needs first to have a physical access. Then he will still have to find the original password from the hash. Because the level of security is high for edge agent, they are considered to be trustable.

Direct authentication to cloud agent

As for the edge agent, the direct authentication to the cloud agent will require the combination of the different factor. In most of the case we use 2-factor authentication using the combination of something you know like a password and something you have like the access to your email address or a device.

The difference here is that you will have to rely on your agent provider methods used to store the secret information they will use to authenticate you to give you access to your agent. A malicious system administrator or an hacker with access to the agent provider environment could compromise your agent if the security of the system is not high enough. That is the reason why the cloud agent is considered to be semi-trustable.

Indirect authentication to cloud agent

One of the solutions to take advantage of both edge and cloud agent is to use them both. We will use the edge agent that is more secure to keep the keys that are used to perform sensitive actions. And you will use a cloud agent to receive communications from other agents anytime, even when the edge agent is off. Since the cloud agent is less secure, the identity owner should keep the most sensitive keys in his edge agent.

To use a cloud agent in combination with an edge agent, the user will need first to authenticate to his edge agent. Then, whenever the user has to go through the cloud agent, a challenge will be sent to the edge agent to make sure you are in control of the key pair associated with the relationship. In the next Section, we will see the key management more in details.

3.3.4 Key management

In Hyperledger Indy, there are five types of keys that an agent need in order to use the different DID methods and start to exchange verifiable credentials:

1. **The link secret:** The link secret is a large random number that only the identity owner knows. The first agent created by an identity owner will create this value, encrypt it and store it within the agent. To request a credentials or to later present proof of a credential about your identity, you must own this secret that is used to determine you are the owner of a credential.

2. **The DID keys:** Each agent has to manage the pair of key for each DID they are in control of in order to sign and verify operations related to DIDs. It is recommended to use one DID per relationship with other identity owners to avoid identifier correlation which can result on a large number of DID keypair to manage.

3. **The agent policy keys:** Each agent also have one key pair. The public key is stored in the agent policy registry of the blockchain that can be used to revoke agents that have been compromised. You can change agent policies by making transactions to the registry using the private key. Verifiers should always check the registry to see if the agent is authorized.

4. **The agent recovery keys:** Agents can use a keypair to recover their wallet in case of loss. The public recovery key (PK1) is stored within the agent and the private key(PrK2) has to be stored offline or divided into shares and distributed to trustees (see Section 3.6.2). To encrypt a backup of a wallet, an ephemeral keypair is created for the encryption. To create the wallet encryption key, we use the ephemeral private key (ePrK4) generated to perform the Diffie-Hellman agreement[43] with the public recovery key (PK1). The ephemeral private key (ePrK4) is forgotten and the public ephemeral key (ePK3) is stored alongside the encrypted wallet. To decrypt the backup, the private

key (PrK2) is used to perform again the Diffie-Hellman agreement with the ephemeral public key (ePK3) to create the same wallet encryption key (see Figure 3.2).

5. **The wallet encryption keys:** The wallet encryption keys are generated using the above procedure. For each new encryption, we generate a new encryption key.

All of these keys are stored inside the encrypted wallet in the device. It is also possible to share these keys to multiple agents in order to use them to perform actions with your self-sovereign identity from different devices.

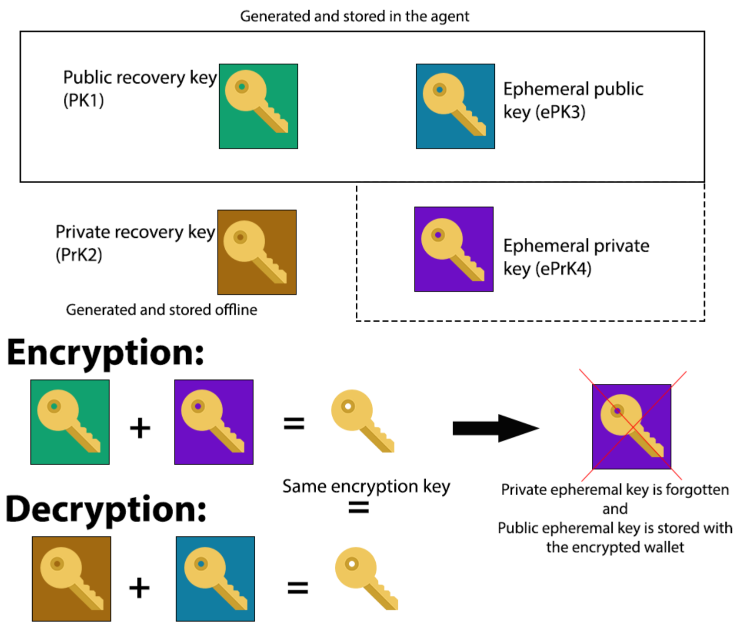

Generated and stored in the agent
Public recovery key (PK1)
Ephemeral public key (ePK3)
Private recovery key (PrK2)
Generated and stored offline
Ephemeral private key (ePrK4)
Encryption:
+
=
Decryption:
+
=
Same encryption key
=
Private epheremal key is forgotten and Public epheremal key is stored with the encrypted wallet

Figure 3.2: Diffie-Hellman agreement for wallet recovery

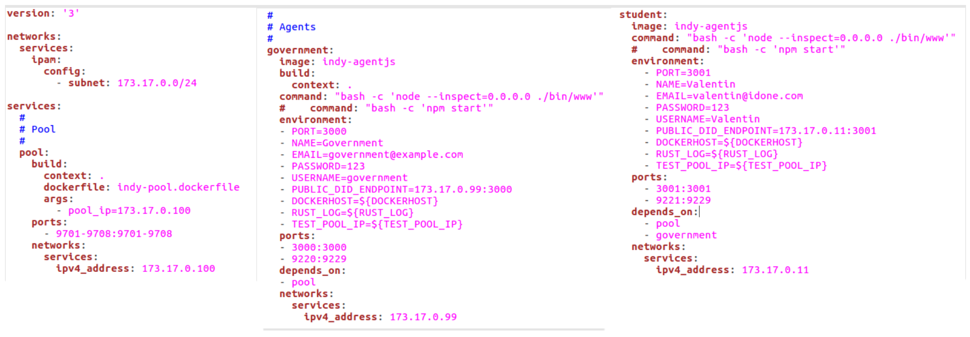
3.4 Agents implementation in educational environment

We mentioned in our design goals that we want to store wallets in both offline and online agents in order to provide security, reliability and availability altogether. However, the implementation of an edge agent with Hyperledger Indy such as mobile application or browser extension like Metamask for Ethereum was not well documented. Therefore, we decided to simplify our architecture by prototyping a system where users can connect themselves to a web service providing them with a cloud agent. To make our tests, we created agent and stored identity wallets and their associated credentials locally on a virtual machine using docker containers. We integrated ”Mobile Connect et Moi” to our agents to authenticate users. This authentication solution is used by OBS and allow french users that suscribed to Orange as their mobile network operator to authenticate themselves to all french public services with their mobile number. Our prototype is designed for physical person but can also be applied to moral entities such as businesses and organizations that can own an identity but can’t use an edge agent by themselves and need to delegate the access to their representative.

In this Section, we first present the method used to deploy our prototype for an educational environment use case where the implementation of self-sovereign identity and the exchange of verifiable credentials makes sense. We give a description of our interface and the Mobile Connect solution that we used to authenticate user. Finally we discuss agent-to-agent protocol that is used to secure communications between agents.

3.4.1 Agent deployment

To deploy the ledger and the four agents present in our architecture, we used docker-compose (Figure 3.3) to configure and run multiple docker container. For our prototype, we run five containers, one for the ledger of Hyperledger indy’s node that was built using an image of a pool of four nodesand deployed on a local network. The other four containers representing the cloud agents of four actors of the educational platform: the government to provide legal identity to students and teachers, the university that need to register students and to deliver deliver degrees, the teacher that need to share information with their students and the student. These four containers are connected to the ledger and were running the nodeJS application we developed and deployed on different ports of the computer local environment.



```yaml
version: '3'

networks:
  services:
    ipam:
      config:
        - subnet: 173.17.0.0/24

services:
  #
  # Pool
  #
  pool:
    build:
      context: .
      dockerfile: indy-pool.dockerfile
      args:
        - pool_ip=173.17.0.100
    ports:
      - 9701-9708:9701-9708
    networks:
      services:
        ipv4_address: 173.17.0.100
```

```yaml
#
# Agents
#
government:
  image: indy-agentjs
  build:
    context: .
  command: "bash -c 'node --inspect=0.0.0.0 ./bin/www'"
  #    command: "bash -c 'npm start'"
  environment:
  - PORT=3000
  - NAME=Government
  - EMAIL=government@example.com
  - PASSWORD=123
  - USERNAME=government
  - PUBLIC_DID_ENDPOINT=173.17.0.99:3000
  - DOCKERHOST=${DOCKERHOST}
  - RUST_LOG=${RUST_LOG}
  - TEST_POOL_IP=${TEST_POOL_IP}
  ports:
  - 3000:3000
  - 9220:9229
  depends_on:
  - pool
  networks:
    services:
      ipv4_address: 173.17.0.99
```

```yaml
student:
  image: indy-agentjs
  command: "bash -c 'node --inspect=0.0.0.0 ./bin/www'"
  #    command: "bash -c 'npm start'"
  environment:
    - PORT=3001
    - NAME=Valentin
    - EMAIL=valentin@idone.com
    - PASSWORD=123
    - USERNAME=Valentin
    - PUBLIC_DID_ENDPOINT=173.17.0.11:3001
    - DOCKERHOST=${DOCKERHOST}
    - RUST_LOG=${RUST_LOG}
    - TEST_POOL_IP=${TEST_POOL_IP}
  ports:
    - 3001:3001
    - 9221:9229
  depends_on:
    - pool
    - government
  networks:
    services:
      ipv4_address: 173.17.0.11
```

Figure 3.3: Extract from the docker-compose YAML file

3.4.2 Interface

When deployed, a simple nodeJS web interface is also generated and include an authentication page and a dashboard where it is possible to perform and monitor actions between the several agents and the ledger we just created. We added a new login option for the login page (Figure 3.4) in order to implement the authentication to the agent via Mobile Connect. Mobile Connect is a strong digital authentication solution provided by the GSMA (GSM Association) to mobile network operators like Orange.

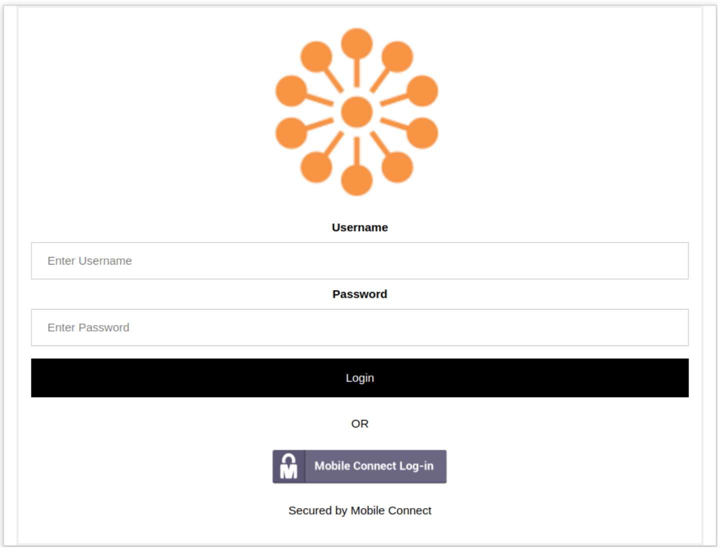


Figure 3.4: Login page

3.4.3 Authentication via Mobile Connect

In the login page that we can see in Figure 3.4, users have two ways to authenticate themselves to their agents. The first one is using a traditional username/password. The second way, Mobile Connect , is more interesting.

Mobile Connect is a simple way for users to connect to an application using their phone number and SIM card[44]. The user creates an account once and he can log into all the websites that integrates the Mobile Connect technology. When he logs in a website, his information is then sent from the network operator to the application provider using token from the open source technology Open ID Connect (see Figure 3.5). In that sense, it is a federated identity. I personally wonder if it makes sense to implement it because it reintroduce centralization in the system. However, it was a strong request from OBS. In the end of the Chapter, we will reflect on the decentralization of the architecture.

We used the Mobile Connect API to implement the authentication within the agent application and I had to create an account on in order to use and test it. To

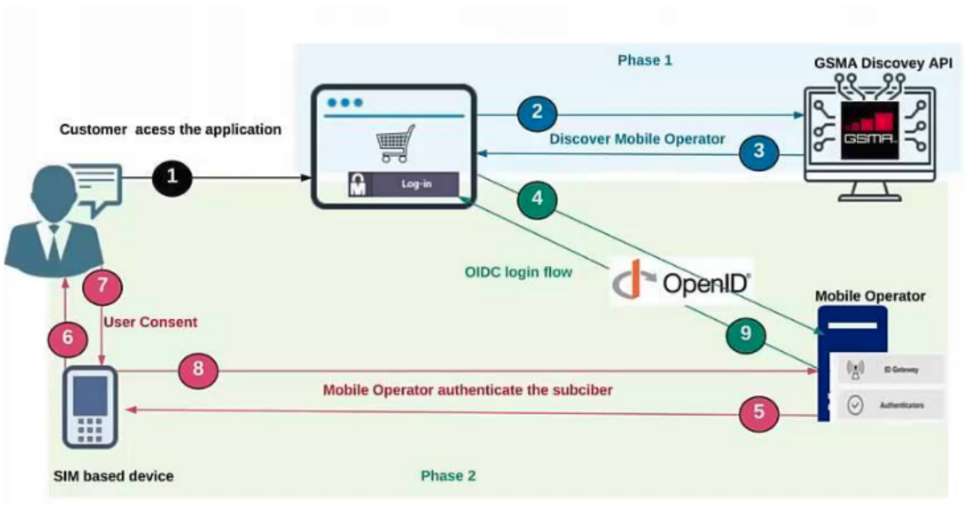


Figure 3.5: Mobile Connect architecture

register, the user need to use his phone number and to scan his identity card with his camera in order to be verified and be trusted by service providers. According to their website, the smartphone application that we used to connect ourselves guarantee the maximum level of assurance (LoA4) against erroneous authentication as it requires the verification of the identity card and the telephone number.

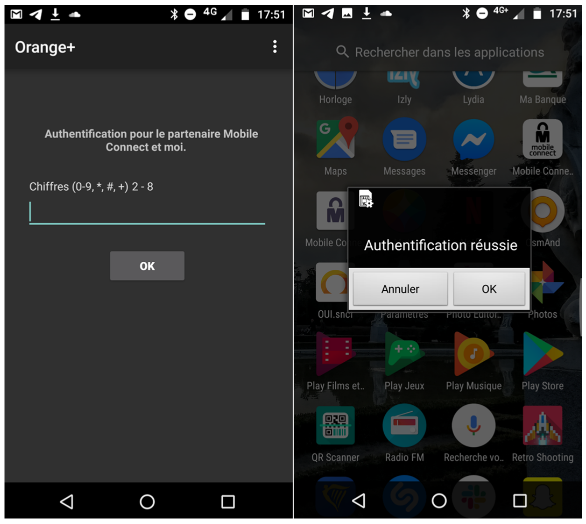
Then when you want to connect to your agent using Mobile Connect, a notification from your SIM card appears on your phone (something the user have) asking you the PIN Code that you registered at the creation of the account (something the user know). The service provider that is in our case the agent provider, receives back a token containing the user PCR (Pseudonymous Customer Reference) and can use this unique identifier to redirect users towards their agent.

3.4.4 Agent-to-agent protocol

After we managed to create and authenticate to agents, we wanted these agents to interact with each others in users behalf. Agents need to connect and maintain relationship, hold, issue and verify credentials. To perform these actions in a secure way agents use cryptographic methods. All these interactions can't be performed by the users and can be described under the agent-to-agent protocol (A2A)[45]. This protocol is used to manage:

- **DID relationships:** Before agents share information safely, they have to create a connection where both agents assign together the keys and the endpoints they will use for authenticated encryption. In Indy, this is done by creating a DID relationship.

  In our prototype, Alice first need the public DID of the university that can be found in the university website or embedded in a QR Code. Then, Alice's


Authentification pour le partenaire Mobile Connect et moi.

Chiffres (0-9, *, #, +) 2 - 8

Authentification réussie

Figure 3.6: PIN code and successful authentication

agent create a key pair and a DID that will be assigned to the University agent. By resolving the public DID, Alice's agent will be able to discover the university endpoint and send an invitation to the University in order to create the relationship.

When the invitation is accepted, university's agent will apply the same process and will create a new DID and a keypair for Alice. It will then notify Alice's agent of the success using the endpoint of Alice's DID.

- **Verifiable credential exchange:** Verifiable credentials exchange between the different identity owners is performed via agent-to-agent communication. After establishing the DID relationship, it is possible to safely request, issue or verify verifiable credentials through a secure channel.

  To issue a credential, an issuer have first to create a credential schema that describes credential attributes list and publish it on the distributed ledger. Then a credential definition that contains the issuer DID, the credential schema as well as secrets used for signing credentials and to revoke it will be created. The private part containing the secret will be kept in the issuer wallet and the public part will be published to the ledger. The issuer can then create an offer for his credentials in order for identity owners to request it. When an identity owner request the credential, the issuer will create and send a credential with different claims to the identity owner. Once the credential is received by the holder it is linked to his master secret so he can later prove to verifiers that is the right owner of the credential. The credential integrity and provenance is also checked on the distributed ledger using the credential definition.[46] If he want to, the verifier can automate is verification using predicates, by doing so it can restrict the credential he is accepting. It is possible for example to use predicates in order to only accept credentials from a whitelist of issuer.

A good thing about the verifiable credential that is built-in Hyperledger Indy and other self-sovereign identity solution in general is the use of selective disclosure and zero-knowledge proofs (ZKPs). In fact, you can select what attribute from your credential you want to disclose as a proof and even prove an information without disclosing at all the underlying data (via ZKP) which is impossible when you use traditional identity documents. Let's take an identity card as an example (see Figure 3.6), on the left you can see a french identity card that can be required to perform an action that requires to have the majority; buying alcohol for example. Even in this example, there is information that I do not disclose because it could be use to usurp my identity. With the selective disclosure the user can select only the information that is necessary. In this case, an identity card delivered by the government and the picture to see check if it's the right person. The seller will also need to know if the buyer is over eighteen but he doesn't need to know the exact birthdate. In this case, we can use the ZKP which use cryptographic techniques to prove that a claim is true whithout revealing its attributes to protect user privacy[47].

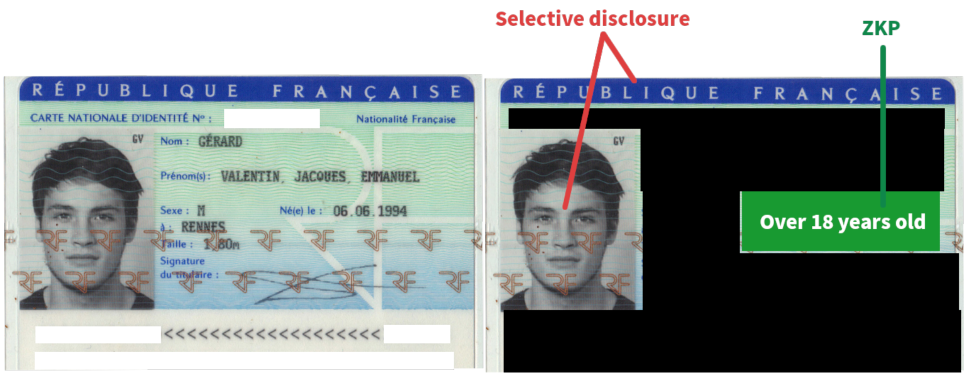


Figure 3.7: Selective disclosure and Zero-Knowledge proof

3.5 General remarks

To conclude this chapter, we give general remarks on the blockchain infrastructure itself. More especially, these remarks focus on the aspects of trust, security and decentralization.

3.5.1 Note on trust

Trust should be guaranteed by our architecture that is based on the self-sovereign identity model to ensure no one can cheat. However, even today, when the technology is supposed to be safe, breaches often appear because of human errors that can be avoided only through education. Moreover, people with little or no technical knowledge won't be able to understand how exactly the architecture can guarantee trust.

More than the technology itself, users need trust the technology providers. In his TedX talk, James Davis defines three drivers of trust[48] that should be taking into account as much as the technology part to build a self-sovereign identity solution that people can trust:

1. **Perceived ability**: Can they?

   Can customers trust our ability to provide them a good service that is respecting their privacy? In this case, we should demonstrate that we have the ability to build complex system with state-of-the-art technologies that helps building trust in online interactions and the ability we have to make it easy to use for the users.

2. **Perceived benevolence**: Do they? Can customers trust us when we say we care about them? In this case, we should demonstrate the benefits of the service to them and provide them with support whenever they need help.

3. **Perceived integrity**: Will they? Does the customers share our values and does they trust our ability to follow it? In this case, we should expose clearly our value to our customers. If they share those values with us and that we stick to it. They should be able to trust us and the technology we developed more easily.

Because people does not necessarily understand how self-sovereign identity help them to control their personal data and improve trust in digital interactions. Any project trying to build self-sovereign identities, even the well-designed one, should take into account these three drivers in order to be adopted by users.

3.5.2 Note on security

There are few security considerations that we have to take into account when we store our identity and important credentials associated to it in an offline or online agent.

Edge agent is lost or doesn't work anymore

If you lose your device or it doesn't work anymore, you would also lose all the important keys needed to manage and use your digital identity such as authenticating as owner of your DID and generate proof for your credentials. In fact if you have no way to recover these keys you would have to contact all your credential's issuers to revoke each one of them and request them again. To counter this problem, two recovery methods are recommended:

- **Offline recovery**: The easiest way is to create an encrypted backup of your identity wallet within your cloud agent and use an removable media like a USB key to store the backup recovery key. It is also possible to write it down on a paper. In both case, the identity owner have to store the offline copy in a safe place as well to remember its location.

- **Social recovery**: Social recovery is an other methods allows the recovery to be accomplished entirely online so the identity owner do not have to worry about the safety of his offline recovery key. In social recovery, you can split your recovery key into shares and distribute shares to trustees who can be person on organizations you trust. This trustees will securely store the share you gave them and give them back to you if you need to recover your wallet. You can then recombine the different shares to obtain your recovery key. To generate the shares and recombine them, the Shamir's Secret Sharing algorithm [49] can be used.

  Each of the trustees can't obtain the recovery key from their key however if they join each other they could recombine their shares to obtain your recovery key. That is the only problem of this methods and that is why you should chose your trustees carefully.

Using an encrypted backup and storing the recovery key with one of those two methods is strongly recommended, and can prevent the definitive lost of the identity owner's wallet.

Cloud agent is compromised

Even though cloud agent are very reliable. However since they are stored online, even if they are supposed to be secure, they are more likely to be subject to attackers. If an attacker manage somehow to steal the keys of your agent, the compromise can be either passive or active. In the first case, the identity owner is not aware that the agent is compromised and is not taking into countermeasure because the attacker did not provide direct evidence of his intrusion . He could just use the key to watch over your different communication. In the second case, the identity owner is aware that is keys have been exposed because the attacker used his identity to commit fraud or locked the owner out of his wallet. In this case, the owner know he have to take actions to get rid of the attacker. There are two mechanisms that can help us to protect our identity in both case:

- **Key rotation**: This is a process that allow an identity owner to periodically change his keys. New ones are created while the previous one expire or are revoked. Is is especially useful in case of passive compromise. By rotating key regularly, you can prevent attackers to use the keys he managed to steal even when you don't know you were compromised. It is also useful to rotate key in order to use up to date algorithms to generate new keys as encryption breaking technologies are constantly advancing.

- **Key revocation**: It is important to be able to revoke keys on demand. Otherwise even though the identity owner know he is actively under attack he would not be able to do anything. Removing the key from the wallet is not enough because it does not prevent the attacker to use it. The key have to be permanently retired.

  In Hyperledger Indy, the cryptographic accumulator method is used to support key revocation[50]. The cryptographic accumulator is a number referenced in a revocation registry within the ledger. This registry is checked instantly by the verifier to know if they key used to generate a proof was revoked or not.

Key rotation and key revocation are standard practices in key management systems. However the decentralized nature of self-sovereign identity make it challenging to implement correctly.

Blockchain is compromised

The blockchain being compromised is the worst case scenario but also the less-likely to happen if the blockchain used to support the self-sovereign identity is picked correctly. The blockchain picked should have counter measure for the most known attacks:

- **The Sybil attack**: When an attacker create many agents within the network in order to use their voting power to gain influence, we speak of a Sybil attack. The poof-of-work consensus algorithm is using peer-review where reviewers(miners) have to consume resources in order to vote was designed to prevent this attack and is used by most blockchain but results in slow transaction speed.

  In trustchain, the problem is tackled differently. A different structure was designed to make the attacker only able to create trust with himself. By doing so the attacker can't get influence over other actors of the network and the profit of the sibyl attack is greatly minimized[51].

- **The 51% majority attack**: When an attacker or a group of attacker has control over the majority of the network mining hash rate. It allows him to manipulate the network has he can mine block faster than any other and use it to perform double spending attack. We talk about double-spending when we make a transaction on the network, waiting for it to be validated and using the majority of hashpower to fork the blockchain at a point prior to the transaction so it does not appear in the ledger even though the transaction has been processed.

  In fact it is very difficult perform such an attack because of the mining cost. To counter this attack, the network has to be as decentralized as possible.

Moreover, it's not in the interest of the miners to group up to get the majority because the trust in the network would be lost and the ecosystem could collapse.

- **The DDoS atack**: We talk about a Direct Denial of Service(DDoS) when an attacker is sending a massive quantities of message towards the network preventing legitimate request to be processed. While the ledger and wallets can't be compromised in a DDoS attack, the network could be very slow or not available at all.

  Still, it's near to impossible to perform a DDoS attack on a blockchain because you would have to do it on every nodes of the network at once.

- **The routing attack**: While the nodes supporting a blockchain are more or less decentralized, there is a kind of attack that is often overlooked. At a lower level, we have Internet Service Providers(ISP) that represents point of failure due to their centralization. For the Bitcoin blockchain for exemple, 3 ISPs are routing 60% of the traffic of the whole network[52]. By intercepting, the traffic sent between ISPs, an attacker could seperate the network in two partitions. When these partitions would join together, the one with the shorter chain would be wiped and all the transactions made on these partitions would disappear. Attackers could take advantage of these attacks to perform double-spending attacks.

3.5.3 Note on decentralization

When we provide a cloud agent as a service, it as the effect to recentralize the architecture, especially when you only have one provider available. What is critical here to avoid recentralization is to have multiple service provider and to promote wallet portability. The user or customer should be able to change of service provider anytime and easily. That is another reason why service providers should agree on some standards for cloud agent to allow the migration from one service to another.

Using Mobile Connect to authenticate the users also recentralizes the identity because we have to rely on a service provider for authentication. However, it is a secure solution that benefit from recovery mechanisms which can be really useful to careless users. Again, to avoid this recentralization, multiple authentication solution should be available so the user is able to chose the one he is trusting the most. It would also avoid the central point of failure issue.

It is still really difficult today to have a true self-sovereign identity where no third parties are involved because the technology is not user friendly yet. Some people and companies might find easier and reassuring to use a service well known in which they trust. Still, a decentralized identity have many advantages and we have to find a trade off for people to be able to benefit from it while always insuring the maximum security and privacy of their identity.

Chapter 4

Credential based access management

In Chapter 3, we have built an agent prototype with Hyperledger Indy that supports self-sovereign identity and the exchange of verifiable credentials. In this Chapter, we integrate a plugin to our prototype where we simulate a workspace that is shared by students, teachers and maintained by a university. For this configuration, we design a solution that allows actors of the academic ecosystem to access a private workspace hosted in a public cloud using the self-sovereign identity they created with the architecture we implemented in the previous chapter. By integrating this plugin, we attempt to discover if it is possible to use verifiable credentials for access management (RQ3).

In Section 4.1, we present an introduction to identity and access management. Then, in Section 4.2, we will describe the ecosystem of the new service and our design goals. Then we use credentials and we define access policies for the different actors of the ecosystem to manage their access to the service in Section 4.3. To conclude, we compare and evaluate access control using verifiable credentials to traditional solutions in Section 4.4.

4.1 Identity and Access Management

Identity and access management are related but they are not similar. Identity management focus on users authentication. To add a user to a system, his identity has to be verified. This is usually done by creating an account for the user inside the organization system with more or less verification such as an email with a verification link or even captures of your identity card for the ones that require more security.

As we have seen in Chapter 3, in self-sovereign identity we don't need these mechanisms. Instead, the authentication aspect of identity is delegated to the agent that hold verifiable credential whose authenticity can be checked on the distributed ledger. If they can trust this ecosystem enough, then service providers or any organizations would not have to take care less about the security of user data and would make considerable savings. Instead, they would just have to get an agent for themselves in order to verify the credentials of users they need to interact with like employees, customers or even machines.

Access management is the step after authentication and is about authorization. Once, a user is authenticated in the system, we can define policies to restrict his access to information or functionalities within the system. In this Section, we explore the different access control methods and explain how we can use credential for access management.

4.1.1 Access control methods

In order to implement access management in a system, several methods has been designed over the years. Here is a list of the main ones[53]:

- **Mandatory access control (MAC):** It is the strictest of all methods. In mandatory access control, only the system administrator can configure the access policies of the resources. In this method, each resource and account on the system has security labels including a classification (public, private, confidential etc.) and a category that can allow the system administrator to give more indication such as the management level or the department name. When an account want to access a resource, we look if the user credentials match with the object security and category labels before to give him the access. This is maybe the most secure method for access control but it is a lot of work to maintain as the system administrator will have to constantly update the security labels of resources as well as the categorization and classification of accounts.

- **Discretionary access control (DAC):** In this method, unlike for the MAC, each user can control the access to their own data. Instead of security labels, each resources is associated to a list of users where the owner of the resources can set the permissions for the different users. This method is more flexible than the previous one and represent less work for the system administrator but also represent more risks as users could configure insecure permissions.

- **Role-based access control (RBAC):** This method is based on the user function within the organization's system. In RBAC, permissions of each resources are assigned to particular roles by the system administrator and all users with the corresponding roles can access to these resources. Roles are assigned to users ansd are hierarchical; they inherit the permission from their parents. This method simplify greatly the access management of the system and is widely used today.

- **Attribute-based access control (ABAC):** In this method, access rights are granted through the verification of different attributes . The policies that are used if the user can access the resource are written in XACML (eXtensive Access Control Markup Language). This language is easy to read and write and allow the system administrator to define policies that take into account different attributes from the user, from the resource or even from the environment(e.g., time or location) at the same time. All these attributes are then processed by an authorization engine that will permit or deny the access to the user. Although ABAC is more difficult to implement, it allows to describe more complex system and make use of dynamic parameters to allow a fine-grained access control. It is often considered as the next-generation access control method.

Access management is a very important aspect of security. These methods have been developed over the years to enhance the confidentiality, integrity and flexibility of access management. However, they were designed for centralized identity ecosystems. Are these methods relevant for self-sovereign-identity? This is the question we are trying to answer in this Chapter.

4.1.2 Credential based access management

In this chapter, we attempt to implement 'different' access control methods using the verifiable credentials of identity owners for verifying their authorization to allow or deny them the access to specific resources. Using self-sovereign identity and verifiable credentials for access management would open up many opportunities. First, there will be no need to create an account for the user in the system of the organization, neither to take care of the security of this account. Then, we could define policies that are taking into account the attributes present in credentials issued by any organizations. The number of combination would then be infinite and would bring access management to another level.

In the next Section, we first describe the plugin integrated within the agent, its ecosystem and the different credentials we will create for the different actors in order to test the different methods.

4.2 Agent plugin description

The goal of the plugin we integrate to our agent is to define different access policies for different users of an academic platform. By doing so, we can restrict the resources of the platform to some users that are holding specific credentials. Our goal is to prove that verifiable credentials can be used within the scope of identity and access management.

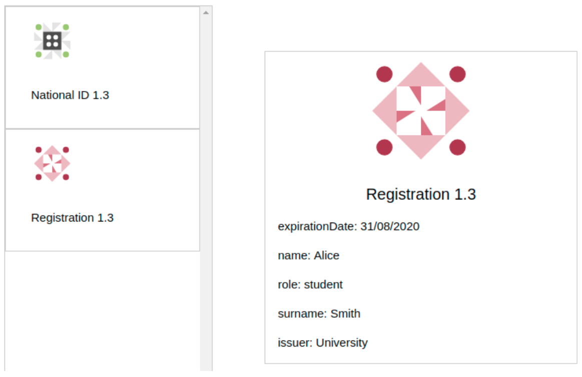
4.2.1 Academic platform ecosystem

The first thing we have to do before to design this platform is to define the different users that will use it. In other words: its ecosystem. In order to keep it simple, we defined three important roles:

- **Students:** They are applying for the program that interest them in the university. If they are admitted they have to register to the university. They have also to register to a set of classes they have or they want to attend. They have to register themselves to pass exams. After their exams, they obtain a degree from the university attesting the completion of their program.
- **Teachers:** They also have to apply for open position within the university. If they are admitted, the administration is registering them within the university. Then they have to prepare and give lectures to students. A platform is also available for them to create repository and share document with students or give them assignments they have to complete online. At the end of the course, they have to give students their final grade.
- **The university:** The university is responsible of the development of the platform and the administration of the platform. It is also in charge of issuing credentials to all actors within the ecosystem (i.e. registration, degree, proof of employment). It is a process that can be automated and can be used to update access level of users of the platform.

4.2.2 Credential's categories and use cases

In this section, we attempt to list the verifiable credentials we need for our platform to define access control for users. We assume that every actors already own their self-sovereign identity.


National ID 1.3

Registration 1.3

Registration 1.3

expirationDate: 31/08/2020

name: Alice

role: student

surname: Smith

issuer: University

Figure 4.1: Credentials details (Registration)

National identity credential

The national identity credential is issued by the government and is used for the first authentication of the user before he can be registered within the system. In our example, it contains its name, surname and the address of the identity owner.

University registration credential

When one of the actors we identified beforehand is admitted or hired in the university, he receives a ”registration credential” containing his basics information as well as his role and the duration of his contract with the university. This first credential will allow actors to access the platform and to display their basic information.

Course registration credential

Teachers will be able to create a workspace for their courses and issue credential to student who have registered in their course. This credential will gave them access to the course repository.

Degree credential

When he validate his year, the student will receive a degree credential that could be required by his future employer. Micro-credentials could also be issued after the completion of each course. These micro-credentials could be used as a condition to get the degree at the end of the year.

Other credentials

We presented the main credentials needed to implement access management in our platform. For a more complex we could make use of more credentials such as:

- **Custom credentials:** By allowing system administrators to create custom credentials, we help them to answer to specific use cases. They could be use for example to attest of special skills of a teacher or an employee in order to verify was trained to perform a certain action. They can be issued by the university or they could be issued by other organizations.
- **External credentials:** As for the nation identity credential issued by the government, we could use other external credentials such as a credential with your banking details issued by your bank.

4.3 Plugin integration

In our plugin, we inspire ourselves from the different access control methods that we discussed in the Section 4.1. By implementing these methods, we want to give the authorization for identity owners to access different resources and different functionalities depending on the credentials they have in their digital wallet.

4.3.1 Architecture

In order to experiment the different access control methods in our service, we added a page inside of our web agent that is connected to a dropbox folder. In the future it would be best to use a browser extension or a mobile wallet like MetaMask/TrustWallet for Ethereum that would act as an agent that could directly interact with the dropbox folder but neither of those is provided by Hyperledger Indy yet. Instead, this new page in the agent could be downloaded as a plugin to interact with the university platform within the agent.

The idea is to authenticate yourself with your agent that holds your credentials and this same agent will be in charge to check your credentials and give you the the right access to the dropbox folders using the dropbox API from the agent plugin.

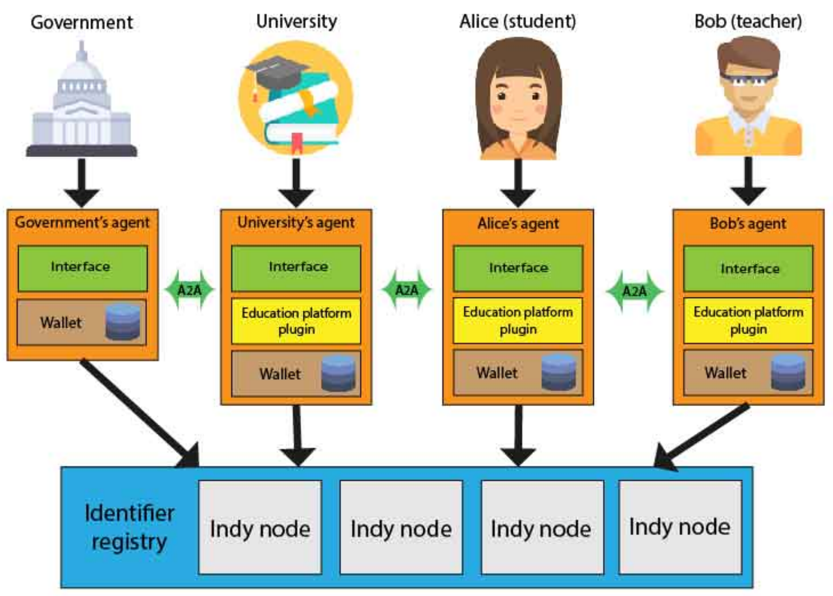
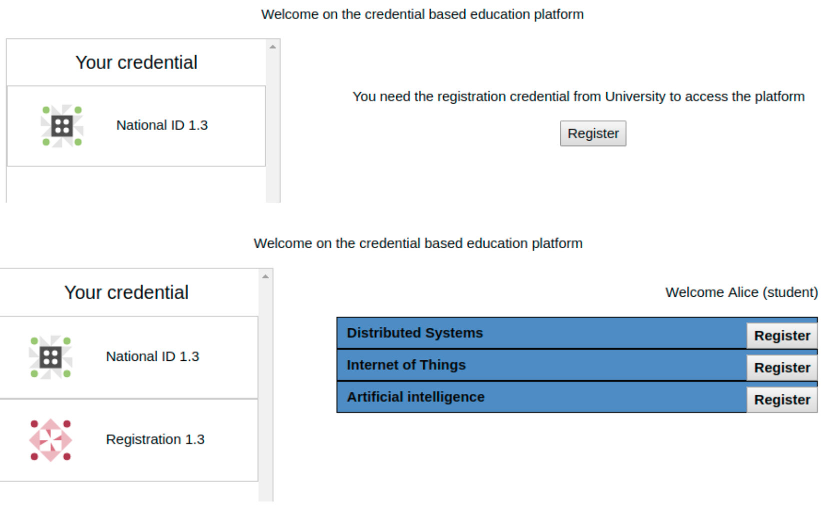


Figure 4.2: Implementation of the education platform as a plugin within our agents

4.3.2 Different access level

In this Subsection, I present the main access level of the platform that I identified and the related credentials that you should hold in your wallet to:

1. **Access to the platform:** To access the educational platform, the user must be registered within the university. If he's not registered, the agent will display a page allowing the user to request the ”university registration” credential (Figure 4.3). In order to issue it the university agent is requiring a claim from a valid ”national identity credential” (predicates). When he receives is credential in his wallet, the will be able to access the dropbox main repository. In this first access use case, we are using mandatory access control (MAC) as only the system administrator can set this rule.


Welcome on the credential based education platform

Your credential

National ID 1.3

You need the registration credential from University to access the platform

Register

Welcome on the credential based education platform

Your credential

National ID 1.3

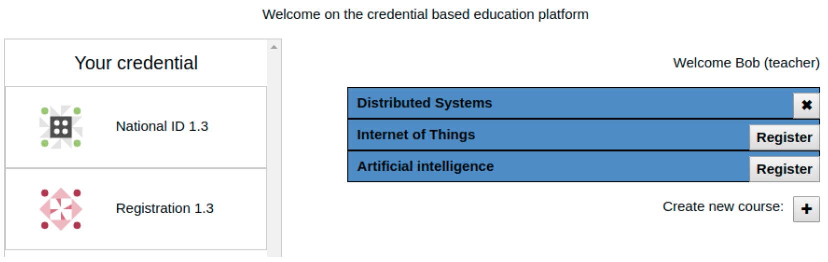
Registration 1.3

Welcome Alice (student)

Distributed Systems	Register
Internet of Things	Register
Artificial intelligence	Register

Figure 4.3: Access to platform resources (1)

2. **Create and access course repository:** Only teachers should be able to create new course repository. For this use case, we are using role-based access control (RBAC) and discretionary access control (DAC), because we want to let teacher manage their own courses. The role is defined in the university registration credential and will let only user with the teacher role create repository in the platform (Figure 4.4). The teacher can then issue credentials to students to give them access to his repository. The students with that credential are able to access the repository. The teacher can revoke the access to the course repository by revoking the credentials. We could imagine adding more claims to that credential in order to allow more complex permissions.


Welcome on the credential based education platform

Your credential

National ID 1.3

Registration 1.3

Welcome Bob (teacher)

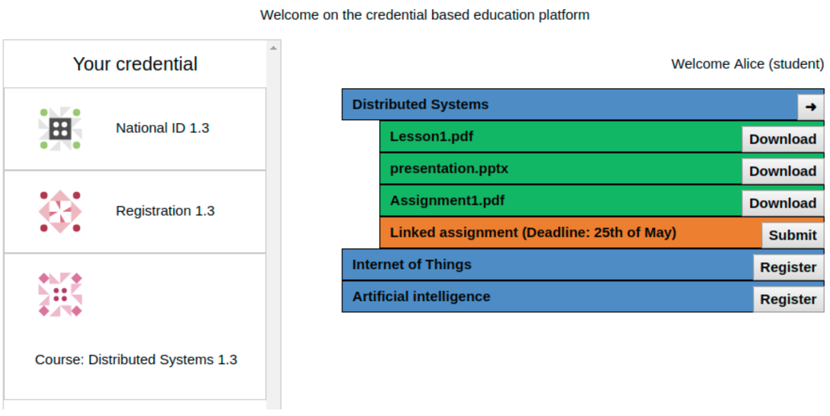
Distributed Systems	✖
Internet of Things	Register
Artificial intelligence	Register

Create new course: +

Figure 4.4: Create course functionality for teacher (2)

3. **Access online assignment:** In order to implement attribute-based access control (ABAC), we imagined an use case where student can access the assignment only in a given period of time. In this example, to access the online assignment, the student must be registered in the course and the date must be

included before the 25th of May (Figure 4.5). When the deadline is passed, the button submit is disabled preventing the access to submission for students.


Welcome on the credential based education platform

Your credential

National ID 1.3

Registration 1.3

Course: Distributed Systems 1.3

Welcome Alice (student)

Distributed Systems

Lesson1.pdf — Download

presentation.pptx — Download

Assignment1.pdf — Download

Linked assignment (Deadline: 25th of May) — Submit

Internet of Things — Register

Artificial intelligence — Register

Figure 4.5: Resource and assignment (3)

4. **Access alumni student network:** At the end of their academic year, students receive their degree as a credential. To show how we can give an utility to this credential within the platform, we make possible for identity owner with this credentials to access the alumni network of the university. This credential can also be used when searching for a job or to access higher education level; when you need to prove that you graduated from the university.

By implementing these use cases using different access control methods, we are trying to show that identity but also access management is possible within the decentralized model of self-sovereign identity. In our implementation, we made it easier to experiment by coding the rules within the agent that allowed us to directly check the credentials in the wallet. In the future, we should have simple way for service providers to check credentials of their users and apply their authorization rules based on user-credentials.

4.4 Performance results

Decentralized ledgers like the blockchain Hyperledger Indy that we use to support our application are known for their performance issues. In this Section, we present the performance results of our implementation. There are five actions that we evaluate (a sequence diagrams of the three last actions can be found in Appendix-Figure 6.1) :

- **Number of write and read per seconds:** We sent a 10,000 write and read transactions to the ledger and calculated the average time of completion.

- **Onboarding time:** The onboarding is the first step to establish a secure connection with the agent of another identity owner. It is performed only one time per relationship. After that, identity owners can exchange messages and credentials without going through this step.
- **Time to get a credential:** The average time for a user to get a credential without passing through the onboarding phase.
- **Time to prove a credential:** The average time needed to prove a credential.

We evaluate these actions with different number of validator nodes. In Hyperledger Indy these nodes are called Stewards and are responsible of the ledger security. For our prototype, we first used a pool of 4 nodes which is the minimum because the Byzantine Fault Tolerance consensus algorithm can handle at best one third of faulty node (f) so the number of nodes should be greater than 3f. That's how we get the formula: N=3f+1. If the ledger is maintained by 4 nodes there cannot be more than one faulty node or the system will collapse. In the same way, there can't be more than 3 faulty nodes in a ledger maintained by 10 nodes.

Now, let's try to add more validator nodes to our pool to see how it affects the performance of our prototype. Theoretically, more there are nodes involved in the maintenance of the ledger more it is fault tolerant and secure. However a bigger number of nodes also affect performance as there is more verification involved. For a good trade-off between security and performance, Hyperledger Indy recommend a pool of 25 nodes. We will try our prototype with 4, 13, 25 and 31 nodes to see if we can verify it. The result from our experiments can be seen in the table 4.1.

	4 nodes	**7 nodes**	**13 nodes**	**25 nodes**	**31 nodes**
write/s	1.014	1.019	1.372	4.471	8.722
read/s	0.025	0.029	0.082	0.093	0.170
onboarding	2.031	2.034	4.287	9.692	16.776
get credential	0.366	0.488	0.613	0.642	1.162
prove credential	0.514	0.613	0.823	0.876	1.611

Table 4.1: Performance results

We can see that the processing time seems to increase when we increase the number of node in each case. It is normal and what we should expect because it becomes more longer to reach a consensus each time you add new nodes to the pool. With 25 nodes, the recommended amount of nodes for security and performance, onboarding a student, verifying his credential and issuing him one would take 11s. However for 1000 students, it would take 160 minutes to onboard them, proving their credentials would take 10mn and issuing them a school certificates 15mn (3 hours in total). If we go with 31 validator nodes this amount of time would jump to 310mn (5 hours). These results are quiet disappointing as universities have usually more than 1000 students enrolled. While students not necessarily register at the same exact time, period like the start of a new school year would create huge congestion on the network. Moreover, using a distributed ledger for only one university makes no sense. It should at least cover all the universities of a country which account to hundred of thousands students.

If we look closer to the read and write per second on the figure 4.6, we can indeed see that the processing time of these two actions for 31 nodes is two times the amount of time for 25 nodes. We think that we should keep 25 nodes as recommended to guarantee both security and performance.

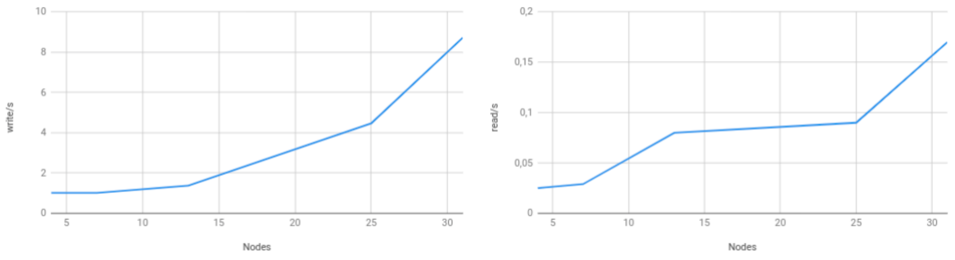


Figure 4.6: Write/s(left) and Read/s(right) per number of nodes

Now, if we look at the onboarding time as well as the time to prove or to get a credential in Figure 4.7. We can see that only the onboarding time suffers from a higher number of nodes. It is because only onboarding require to write on the ledger (2 writes) and only the write operation require to find a consensus between nodes. To verify a credential only require to read the ledger. It is the same to get a credential where the student read the ledger to find the credential schema to create his own. If we consider that every students are already onboarded then the time to verify or to issue 1000 credentials to students is not so bad (10 to 15mn). But as stated earlier, to be useful this solution should include multiple universities with much more than 1000 students.

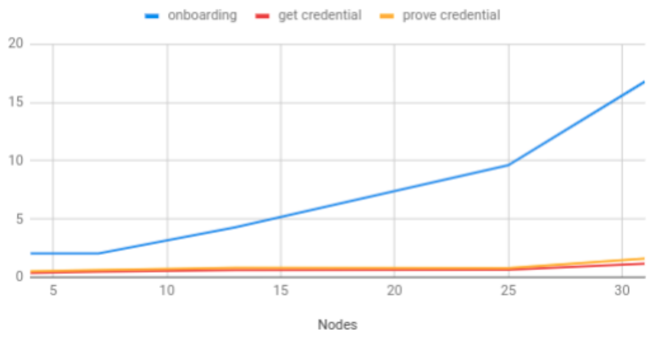


Figure 4.7: Onboarding, issuing and proving a credential per number of nodes

As it is today, we can conclude regarding the results of our experiments that this solution is not scalable and not ready at all for large adoption. However, this prototype still gave us an overview of the capabilities of such a system. With this model, user can have true ownership of their certificate and personal information. They can use these certificate to prove their identity to other identity owners and disclose only what is necessary (via selective disclosure and zero-knowledge proofs). Finally, verifiable credentials can be used to manage users access rights and give instant access to specific resources without any administrator intervention.

Chapter 5

Conclusions and Future Work

As we are interacting online more today than ever before. The number of actors we are interacting with greatly increased too. In a day you can interact with friends, colleagues, strangers, private companies that provide services or product and also with public services. We currently are using identity models that are mostly centralized or federated at best. This models does not allow to manage digital identity in an optimal way. First because, users have to rely on the systems of third-parties and have no view on the way their data is used and if it is safe. Secondly because it results in a bad user experience where the user have to create dozens of different accounts to interact online with different actors and finally because we have no simple way to store, use and verify signed document and certificates because it is difficult to link them to a single digital identity. A user-centric model of identity would partly solve these problems. Some technologies such as OpenID and Facebook Connect allowing users to log in websites that integrated their solutions using a single account exists. However, users still have to create their identities via organizations which for most of the case do not encrypt our communications with other users. Relying on central authorities with no data protection is risky as these organizations could use your identity in a bad way without you noticing or be hacked and get in the hand of potentially dangerous actors. Then we can first conclude, that the trust we can have in today digital communications is not enough.

In this chapter, we first summarize the findings regarding the research questions about the trustworthiness, the security and the management of digital identity using the self-sovereign model as well as its use in access management in Section 5.1. Then, we propose directions for improvements and future work in Section 5.2.

5.1 Conclusions

Our conclusion includes the answers we found to our research questions as well as general remarks for each of them.

RQ1: *Can self-sovereign identities improve trust in digital interactions?*

When using self-sovereign identities, we don't have to rely on a single centralized third-party to create our identity and it makes all the difference. To replace it, we use a decentralized ledger secured by a consensus algorithm that make cheating so difficult that it become possible for strangers to trust each other without any need for central authorities that a majority of people consider trustable enough. The self-sovereign identity model also attempt to create a digital identity following principles that guarantee the best experience for users as well as privacy and protection. To achieve it, this model provide every users with digital wallets that they can use to store credentials and to interact with the ecosystem of people and organizations using the same technology. These wallets are secured by cryptographic mechanisms and the credentials they can hold have a great potential for Identity and Access Management. They are stored within user devices, in a software component called agent that perform encryption and decryption methods in their behalf and allow the secure communication with other identity owners.

Even if the technology is at an early stage and there is still a lot of work to be done to see a large adoption in the coming years, we can definitely say that self-sovereign identities are designed in the best interest of both users and organizations and that this model can improve trust in digital interactions.

RQ2: *How can self-sovereign identities owners manage their identity with an agent, how do they access it and are they safe to use?*

In self-sovereign identity, the agent is the representative of the user. It performs some action such as storing cryptographic keys and user information as verifiable credentials. It also acts as a medium between the decentralized ledger and agents from other self-sovereign identities owners of the ecosystem. Moreover, the agent make sure the information communicated is secured through encryption/decryption mechanisms. It is a very important component of the self sovereign identity model. That is why in RQ2 we asked ourselves how do users access it and how do they manage their identity through it to discover if they were safe to use.

We discovered that agent are composed of three main components, a wallet that store the cryptographic keys and credentials, a set of protocol and methods to secure communications and an interface that is used by the user to manage its identity. Several authentication methods are available to access the agent, to be considered safe, the authentication to the agent should at least verify 2 factors of the following: something the user know (like a password), something the user have (like a device) and something the user is (i.e. biometrics). Because the identity owner is in control of his keys and personal information, he should be careful and try not to lose it. Just in case, it is possible to use recovery methods such as offline recovery where an encrypted backup is stored online while the key to decrypt it is kept safely in an offline medium. If he is not confident in his ability to store the key safely he can use a second method called the social sharing where the key is divided in pieces and shared among people he trusts.

RQ3: *Can verifiable credentials be used to manage users access rights?*
To answer this question, we looked at already existing access control methods. Taking inspiration from these methods we implemented a mix of it such as mandatory, role based and attribute-based access control. We developed a plugin inside the agent and defined policies to use attributes from users credentials as well as environment variables in order to authorize or to deny the access of users to resources of an academic dropbox repository.

Even though the performance results were disappointing (more than 2 hours to onboard 1000 people), we can still conclude that verifiable credentials can indeed be used as a solution for organizations to manage access rights of their users and customers to specific resources. The added value of access management in SSI and verifiable credentials compared to centralized model is that credentials are really owned by the user and not just stored in the organization system. Moreover, it allows organizations to verify the authenticity of the information given by users and onboard them more easily. It is also providing a better experience to users as it takes few seconds to register to services and get access as soon as they receive the access credential in their wallet. Moreover the use of selective disclosure and zero-knowledge proofs allow the verification of claims while preserving the privacy of users. By doing so, user experience is enhanced and organizations do not have to collect data that they don't need about their users.

5.2 Future Work

Our work was a first approach to self-sovereign identity and we obtained valuable insights about the underlying technology and its protocols. However, there is still many different way we can explore further the subject:

- We dedicated this thesis to human identity but humans are not the only one who could benefit from such a technology. Objects are now connected with us on the Internet and their is many use cases where providing them with an identity would be useful. As for humans, it would allow objects to authenticate themselves and be authorized to perform actions regarding of the credentials they have. It would be interesting to go deeper in that directions to see if the protocols we used for human identity would be relevant for the Internet-of-Things ecosystem.
- The objective of this master thesis was to create an agent service to provide user with self-sovereign identities. To create this service, we researched the different protocol that are used to store an identity safely online as well as authenticating its identity owner and authorizing him to access specific resources depending of his credentials. Now that we discovered how to implement it, it would be interesting to see if it's possible to reproduce the authorization part with different existing solutions and to compare their performance.

Bibliography

[1] R.I.M. Dunbar. Neocortex size as a constraint on group size in primates. *Journal of Human Evolution*, 22(6):469 – 493, 1992.

[2] David Kahn. *The codebreakers: the comprehensive history of secret communication from ancient times to the Internet.* Scribners and Sons, 1997.

[3] Ken Jordan, Jan Hauser, and Steven Foster. The augmented social network: Building identity and trust into the next-generation internet, 04 2003.

[4] David Recordon and Drummond Reed. Openid 2.0: a platform for user-centric identity management. pages 11–16, 01 2006.

[5] Djuri Baars. Towards self-sovereign identity using blockchain technology. PhD thesis, Universtiy of Twente, 2016.

[6] European comission. The gdpr: new opportunities, new obligations. *https://ec.europa.eu/commission/sites/beta-political/files/data-protection-factsheet-sme-obligations_en.pdf*, 2018.

[7] The Linux Foundation. Hyperledger indy project homepage. *https://www.hyperledger.org/projects/hyperledger-indy*, 2019.

[8] Paul Payam Almasi. The identity revolution self sovereign powered by blockchain. *https://blog.goodaudience.com/how-blockchain-could-become-the-onramp-towards-self-sovereign-identity-dd234a0ea2a3*, 2018.

[9] Commission Nationale de l'Informatique et des Liberts. Rnipp : Rpertoire national didentification des personnes physiques. *https://www.cnil.fr/fr/rnipp-repertoire-national-didentification-des-personnes-physiques-0*, 2009.

[10] Kevin Featherly. Advanced research projects agency network. *https://www.britannica.com/topic/ARPANET*, 2016.

[11] Cricket Liu and Paul Albitz. *DNS and BIND*. O'Reilly Media, 3 edition, 1998.

[12] Steve DelBianco and Braden Cox. Icann internet governance: Is it working. *Global Business & Development Law Journal*, 21(1):27, 2008.

[13] Whitfield Diffie. The first ten years of public-key cryptography. *Proceedings of the IEEE*, 76(5):560–577, 1988.

[14] Carlisle Adams and Steve Lloyd. *Understanding PKI: concepts, standards, and deployment considerations*. Addison-Wesley Professional, 2003.

[15] Eric Rescorla and A Schiffman. The secure hypertext transfer protocol. Technical report, 1999.

[16] Arun Vishwanath and Hao Chen. Personal communication technologies as an extension of the self: A cross-cultural comparison of people's associations with technology and their symbolic proximity with others. *Journal of the American Society for Information Science and Technology*, 59(11):1761–1775, 2008.

[17] Serge Egelman. My profile is my password, verify me!: the privacy/convenience tradeoff of facebook connect. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2369–2378. ACM, 2013.

[18] Jason Silverstein for CBS News. Hundreds of millions of facebook user records were exposed on amazon cloud server. *https://www.cbsnews.com/news/millions-facebook-user-records-exposed-amazon-cloud-server/*, 2019.

[19] Nik Milanovic for TechCrunch. The next revolution will be reclaiming your digital identity. *https://tcrn.ch/2xM5lQm*, 2017.

[20] Jon Callas and Phillip Zimmermann. The pgp paradigm. *https://github.com/WebOfTrustInfo/rwot1-sf/blob/master/topics-and-advance-readings/PGP-Paradigm.pdf*, 2015.

[21] Christopher Allen. The path to self-sovereign identity. *http://www.lifewithalacrity.com/2016/04/the-path-to-self-soverereign-identity.html*, 2016.

[22] Rebooting the web of trust. Decentralized public key infrastructure. *https://github.com/WebOfTrustInfo/rwot1-sf/blob/master/final-documents/dpki.pdf*, 2015.

[23] Hilarie Orman. Blockchain: The emperors new pki? *IEEE Internet Computing*, 22(2):23–28, 2018.

[24] Black John. Developments in data security breach liability. *The Business Lawyer*, 69(1):199–207, 2013.

[25] Satoshi Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system. 2008.

[26] Iuon-Chang Lin and Tzu-Chun Liao. A survey of blockchain security issues and challenges. *IJ Network Security*, 19(5):653–659, 2017.

[27] Drumond Reed and Manu Sporny. Decentralized identifiers (dids) v0.12. Draft community group report, W3C, March 2019. https://w3c-ccg.github.io/did-spec/.

[28] P Saint-Andre and J Klensin. Uniform resource names (urns). Technical report, 2017.

[29] Manu Sporny and Markus Lanthaler. Json-ld 1.1. Draft community group report, W3C, September 2018. https://json-ld.org/spec/latest/json-ld/.

[30] J. Martin. *Managing the Data Base Environment*. A James Martin book. Pearson Education, Limited, 1983.

[31] Bryce "Zooko" Wilcox. Names: Decentralized, secure, human-meaningful: Choose two. *https://web.archive.org/web/20120204172516/http://zooko.com/distnames.html*, 2001.

[32] Stijn Meijer Tommy Koens. Matching identity management solutions to self-sovereign identity principles. *https://www.slideshare.net/TommyKoens/matching-identity-management-solutions-to-selfsovereign-identity-principles/1*, 2018.

[33] Evernym. Evernym's homepage. *https://www.evernym.com/*, 2019.

[34] uPort. About section. *https://www.uport.me/\#about*, 2019.

[35] Quinten Stokkink and Johan Pouwelse. Deployment of a blockchain-based self-sovereign identity. *CoRR*, abs/1806.01926, 2018.

[36] TU Delft Blockchain-Lab. Trustworthy identity on your phone. *https://www.blockchain-lab.org/trust/*, 2018.

[37] PESTLEAnalysis.com. What is pestle analysis? *ttps://pestleanalysis.com/what-is-pestle-analysis/*, 2015.

[38] E-estonia. E-identity. *https://e-estonia.com/solutions/e-identity/id-card/*, 2001.

[39] European comission. Discover eidas. *https://ec.europa.eu/digital-single-market/en/discover-eidas*, 2018.

[40] Huffpost. Bitcoins carbon footprint as large as las vegas, researchers say. *https://www.huffpost.com/entry/bitcoin-carbon-footprint_n_5d027b25e4b0304a120baedd*, 2019.

[41] Vivien Quma Pierre-Louis Aublin, Sonia Ben Mokhtar. Rbft: Redundant byzantine fault tolerance. *https://pakupaku.me/plaublin/rbft/5000a297.pdf*, 2013.

[42] Daniel Hardmann. 0002: Agents. *https://github.com/hyperledger/indy-hipe/blob/31df09b3949021d790ebc364d7da1b9347821d87/text/0002-agents/README.md*, 2017.

[43] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[44] GSMA. About mobile connect. *https://developer.mobileconnect.io/about*, 2018.

[45] Daniel Hardman. 0003: A2a. *https://github.com/dhh1128/indy-hipe/blob/a2a/text/0003-a2a/README.md*, 2014.

[46] Hyperledger Indy. Anoncreds design. *https://hyperledger-indy.readthedocs.io/projects/sdk/en/latest/docs/design/002-anoncreds/README.html*, 2018.

[47] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge proofs of identity. *Journal of cryptology*, 1(2):77–94, 1988.

[48] James Davis. Building trust - james davis - tedxusu. *https://youtu.be/s9FBK4eprmA*, 2014.

[49] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.

[50] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In *International Workshop on Public Key Cryptography*, pages 481–500. Springer, 2009.

[51] Pim Otte, Martijn de Vos, and Johan Pouwelse. Trustchain: A sybil-resistant scalable blockchain. *Future Generation Computer Systems*, 2017.

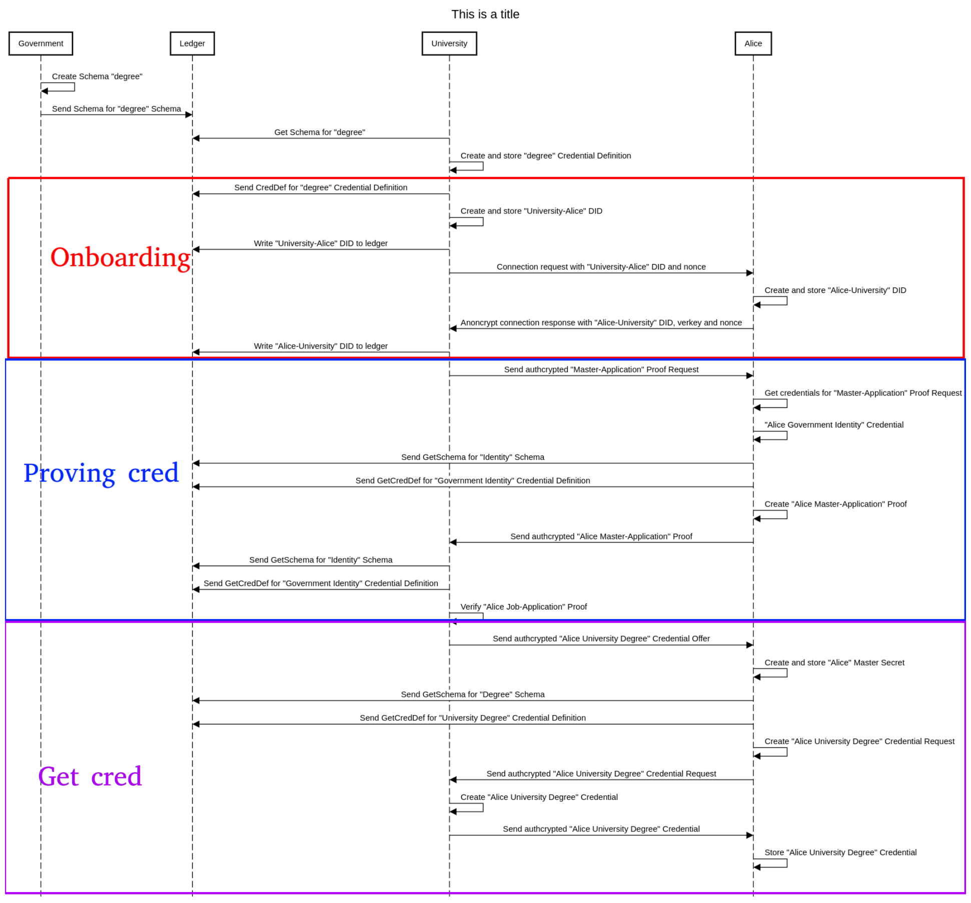
[52] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. Hijacking bitcoin: Routing attacks on cryptocurrencies. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 375–392. IEEE, 2017.

[53] Romuald Thion. Access control models. In *Cyber warfare and cyber terrorism*, pages 318–326. IGI Global, 2007.

Chapter 6

Appendix

The appendix includes a number of resources that we avoided to include in the main chapter to keep the thesis readable. Those resources should bring more details and understanding to the corresponding sections.



This is a title
Government
Ledger
University
Alice
Create Schema "degree"
Send Schema for "degree" Schema
Get Schema for "degree"
Create and store "degree" Credential Definition
Onboarding
Send CredDef for "degree" Credential Definition
Create and store "University-Alice" DID
Write "University-Alice" DID to ledger
Connection request with "University-Alice" DID and nonce
Create and store "Alice-University" DID
Anoncrypt connection response with "Alice-University" DID, verkey and nonce
Write "Alice-University" DID to ledger
Proving cred
Send authcrypted "Master-Application" Proof Request
Get credentials for "Master-Application" Proof Request
"Alice Government Identity" Credential
Send GetSchema for "Identity" Schema
Send GetCredDef for "Government Identity" Credential Definition
Create "Alice Master-Application" Proof
Send authcrypted "Alice Master-Application" Proof
Send GetSchema for "Identity" Schema
Send GetCredDef for "Government Identity" Credential Definition
Verify "Alice Job-Application" Proof
Get cred
Send authcrypted "Alice University Degree" Credential Offer
Create and store "Alice" Master Secret
Send GetSchema for "Degree" Schema
Send GetCredDef for "University Degree" Credential Definition
Create "Alice University Degree" Credential Request
Send authcrypted "Alice University Degree" Credential Request
Create "Alice University Degree" Credential
Send authcrypted "Alice University Degree" Credential
Store "Alice University Degree" Credential

Figure 6.1: Sequence diagram of onboarding, credential issuance and verification