Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Self-Sovereign Identities in Cardossier

Master Thesis

Remo Glauser

glauserr@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Prof. Dr Roger Wattenhofer
Dr Remo Meier, Adnovum Informatik AG

June 11, 2019

Acknowledgements

At this point, I would like to thank all the people who supported this project and shared their expertise. The project could be realized due to the Distributed Computing Group (DISCO) headed by Prof. Dr Roger Wattenhofer, who gave me the opportunity to write this thesis at a project of the Adnovum Informatik AG. Many thanks to my supervisor Dr Remo Meier, who always had valuable advice for me. I thank Matthias Loepfe and Moritz Kuhn, which are in charge of the Cardossier project at Adnovum, for paving the way to this thesis. A thank to the Cardossier team who promptly responded to my questions.

Abstract

Distributed ledger based ecosystems, which go beyond traditional cryptocurrencies, request for privacy-preserving identity management solution. Cardossier is an ecosystem which stores transparently and securely all data form a car's life cycle on an R3 Corda permissioned distributed ledger. It involves all industries of the car ecosystem and allows business processes between the participants. A primary concern of ecosystems like Cardossier is how to manage the data and identity of end-users. We present an architectural solution to identity management for Cardossier based on the concept of self-sovereign identities. It completely decentralizes identity management by letting users control their identifiers and personal data. Our solution enables minimal disclosure of personal information, avoids identity correlation attacks and allows users to give consent to Corda transactions. For this solution, partial elements were implemented as proof-of-work next to the existing Cardossier application.

Contents

Glossary

Agent (Cardossier) A role a node operator can perform by offering identity owner specific services such as Cardossier network access or data storage.

Claim An assertion made about a subject.

Credential A set of one or more claims made by an issuer.

Decentralized identifier A portable URL-based identifier. E.g. did:example:123456abcdef.

Decentralized identifier document A document that is accessible using an verifiable data registry and contains information related to a specific decentralized identifier, such as the associated repository and public key information.

DID owner An entity which owns the DID by being in possession of the private key counterpart of the public keys in the DID Document.

Entity A thing with distinct and independent existence, such as a person, organization, concept, or device.

Holder A role an entity can perform by possessing one or more verifiable credentials.

Identifier A label that uniquely identifies an identity.

Identity A digital representation of an entity. It consists of claims and identifiers.

Identity owner (Cardossier) A role an entity can perform by possessing identifiers and verifiable credentials, which allow to create verifiable presentations.

Issuer A role an entity can perform by asserting claims about one or more subjects, creating a verifiable credential from these claims, and transmitting the verifiable credential to a holder.

Issuer (Cardossier) A role a node operator can perform by issue verifiable credential for an identity owner.

Login credential Identifier plus secret used for authentication.

Node operator (Cardossier) A role a legal entity or person can perform by running Cardossier node and participating on the network.

Subject An entity about which claims are made.

Verifiable credential A W3C standard credential which can be use to build verifiable presentations.

Verifiable presentation A W3C standard credential which can be use to build verifiable presentations.

Verifier A role an entity can perform by receiving one or more verifiable presentations for processing.

Verifier (Cardossier) A role a node operator can perform by receiving and validating verifiable presentations.

Wallet A wallet holding verifiable credentials, storing key materials and enabling to create verifiable presentation.

Acronyms

CA Certificate Authority.

DApp Decentralized Application.

DID Decentralized Identifier.

DID Doc DID Document.

DIF Decentralized Identity Foundation.

DL Distributed Ledger.

DLT Distributed Ledger Technology.

DNS Domain Name Service.

DPKI Decentralized Public Key Infrastructure.

PKI Public Key Infrastructure.

SDK Software Development Kit.

SSI Self-Sovereign Identity.

UTXO Unspent Transaction Output.

VC Verifiable Credential.

VP Verifiable Presentation.

ZK Zero-Knowledge.

ZKP Zero-Knowledge Proof.

CHAPTER 1

Introduction

In recent years distributed ledgers (DLs) became popular due to the rise of the digital currencies like Bitcoin. However, the technology hold potential for applications far beyond Bitcoin and cryptocurrencies. Today, distributed ledgers appear in a variety of commercial applications [14] and are not limited to the financial sector anymore [18]. Such an application is Cardossier [43], a DL-based digital dossier, where all data from a vehicle's life cycle can be stored transparently and securely. An initial focus lies on the used car market that is characterized by a lack of trust, isolated parties and uncertainty, as many parties are involved. The goal of Cardossier is to provide improvements in trust, transparency, efficiency and enable new business opportunities. The application relies on R3 Corda as permissioned DL implementation that allows steering the data flow to parties only involved in business processes. One of the primary concerns of ecosystems like Cardossier is how to manage the data and identity of end-users in a privacy-preserving manner. How can users take control of their identity and interact with the Cardossier ecosystem with privacy and trust? To answer this question, we investigate the concept of self-sovereign identity [13] and its current implementations. We design and prototype a self-sovereign identity management solution for Cardossier.

Document structure: In this chapter, we introduce identity management models, R3 Corda and Cardossier, moreover, we state the detailed problem statement. In Chapter 2, we discuss emerging specifications and published concepts for SSI and related projects. Chapter 3 presents the requirements on identity management, and we evaluate related projects based on them. We demonstrate a concept and architecture in Chapter 4 and a prototype implementation in Chapter 5. Finally, we present our conclusion Chapter 6.

Chapter structure: In the first section, the context of the term "identity" is defined. We differentiate it from other research fields and set it into the context of digital systems (Section 1.1). In the next section, different identity management models from isolated to federated until self-sovereign identities, including their historical appearing, are discussed (Section 1.2). Next, a high-level introduction to the distributed ledger platform R3 Corda is given (Section 1.3). Corda is the building element of the Cardossier project, which is introduced in the subsequent Section 1.4. Finally, the problem statement of this work is stated out and its scope defined in Section 1.5.

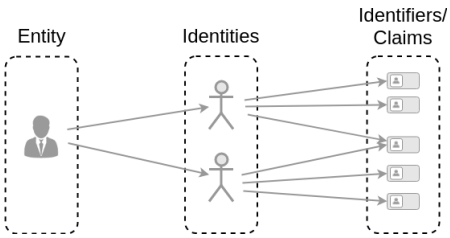

Figure 1.1: Relationship between entity, identity and claims/identifiers

1.1 Identity

According to the ISO/IEC 24760-1 standard, each object with a recognizable, distinct existence such as a car, a person, an organization, a device or a group of such objects is described as an entity. Thus, an entity is an item in the physical and digital world, and it has distinct attributes. For instance, a person who has his name, date of birth or hair colour as an attribute. An identity is defined as a representation of an entity in information and communication technology systems [60]. In this context, identity refers to digital identity and differs from the psychological, sociological and philosophical perspective [40]. Throughout this document, the term identity is used, meaning a digital identity.

In some works, identity is defined as a set of attributes or characteristics [4], [60]. The term claim is used instead, which provides an extended meaning. A claim is a statement about an entity, which can be self-asserted, asserted by an authority or even by a third party. Claims, which are used as unique identifies inside or outside a domain, are specified as identifiers. In Figure 1.1, the relationship between entity, identity and claims/identifiers is illustrated. The figure shows that an entity may have multiple identities, and each identity may consist of multiple claims and identifiers. Further, it points out that identity does not represent more than one single entity. People may argue that a couple sharing a bank account are two entities (two persons) having the same identity. It is not the case from a system point of view since the couple itself is an entity, and the persons are attributes of it.

1.2 Identity Management Models

Since the beginning of computer systems and networks, the question of adequate identity management arise. In early days when the systems have been less complex, rather primitive or only a few people had access to such systems, simple identity management models like the isolated model or the centralized model have been implemented, e.g. IP addresses and domain names determined by the organization IANA. Later on, with the rise of commercial web services and wide adoption of the Internet, Microsoft introduced a federated identity system with Mircosoft Passport in 1999 [3]. Social networks emerged, and further companies like Facebook, Linkedin or Google started to offer federated identities [39]. Since the user's amount of login credentials increase with each identity, user-centric identity [4] solutions

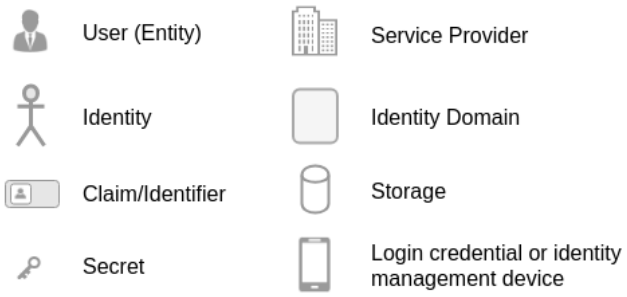

Figure 1.2: Legend of identity model illustrations

have been presented, such as password managers. Finally, With the emergence of blockchain technology, self-sovereign identities have been proposed [13].

1.2.1 Isolated Identity

The service provider has full administrative control over the identities by managing them in his registry. A user requesting service for the first time is enforced to go through a registration process. The registration process can have different elements, for instance, passport based identification or verification of attributes such as e-mail address or phone number. However, it keeps in common that the service provider creates an identity of the users inside its domain.

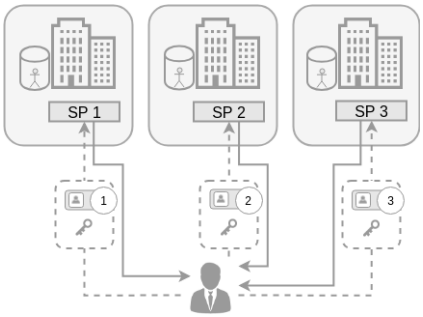

Figure 1.3: Isolated Identity: An identity for each domain.
(Legend: Figure 1.2)

Figure 1.3 illustrates the model and describes an authentication process. All service providers are isolated in their domain and manage the identities of the user separately. The user possesses different login credentials (identifier and secret) for each provider for authentication and accessing the service.

The usability for the user decreases with each new service he is going to use. Therefore, manually managing all login credentials becomes challenging.

1.2.2 Centralized Identity

Besides the isolated model, the identities controlled by a single authority is still the most spread identity management approach in the digital world.

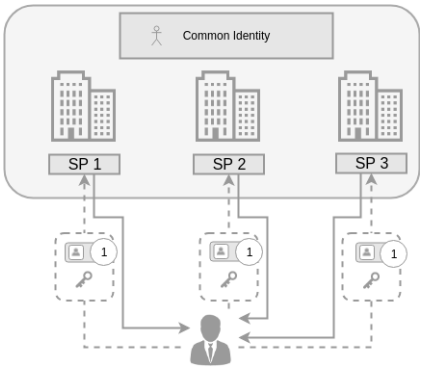

Figure 1.4: Centralized Identity: An identity for one single domain containing all organizations.
(Legend: Figure 1.2)

The centralized model is illustrated in Figure 1.4. It shows that a user has only one login credentials, and thus, he is represented by one single identity for all services in the domain. The common identity is created and managed by a central authority, which acts as an identity provider.

It was the first model implemented straight away from the beginning of the digitization. In the Internet's early days, organizations like IANA became validator of IP addresses, and ICANN placed itself as the central authority of the Domain Name Service (DNS). Later on, with the rise of commercial sites on the Internet certificate authorities (CAs) appeared to prove ownership of the web pages. Some of these organizations stepped to hierarchical models. Nevertheless, they remained centralized at the root controller until today [13].

1.2.3 Federated Identity

In the federated model, digital identity is not limited to one single domain. Instead, users can utilize the same identity on several service providers, and thus, they can move from service to service inside the federation. Nevertheless, each site remained an authority, which holds control on the identities [13].

Figure 1.5 illustrates a federation. The providers cooperate by exchanging the identities between each other. It allows a user to login in with a single login credential to access several distinct services, called single sign-on.

Prominent examples of federated identity platforms are Google account, Microsoft account or LinkedIn [39], which allow authentication to third party websites. However, privacy protection is a crucial issue of such systems [7], since it allows to collect data about the user's activities to the identity provider.

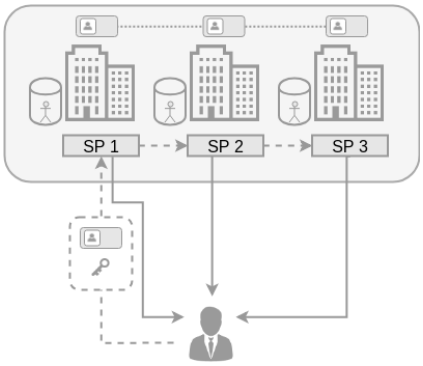

Figure 1.5: Federated Identity: An identity shared on request inside a single domain
(Legend: Figure 1.2)

1.2.4 User-Centric Identity

From a user perspective, an increasing number of login credentials rapidly becomes unmanageable. If the usability is poor, then the authentication itself will be weak since the users are

not able to handle a large number of credentials adequately. The issue is considered by the federated model, which has been motivated by the need to simplify the user experience. The idea is that a user requires only a single login credential, which would make memorizing or storing primarily acceptable. However, the model is only effective if one or only a few identity domain's exist, which is hardly conceivable. There will never be a single identity domain for all service providers. Further, it requires different level of security and risk for different mechanisms, infrastructures and credentials [4].

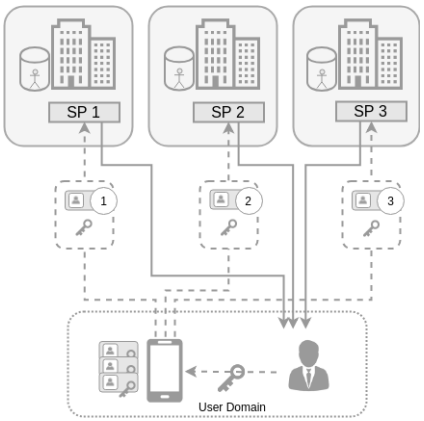

Figure 1.6: User-centric Identity: A isolated identity with client side management tool
(Legend: Figure 1.2)

The user-centric model intends to increase the usability drastically by establishing an automated login credential management on the user side. Figure 1.6 illustrates the model. The service providers are isolated, and they run their identity domain as in the isolated model. For the user, there is a significant change that he is equipped with a personal credential management device. Thus, only one single secret for unlocking the device is required to be memorized by the user. Besides better usability, stronger secrets can be applied for authentication to a service provider [4].

However, the model poses further challenges in terms of security and data loss. In case the credential management device becomes compromised, the malicious party obtains access to all services and potentially private data of the user. Furthermore, an adequate data recovery mechanism is crucial since it has to be consider that the credential management device can be lost.

1.2.5 Self-Sovereign Identity (SSI)

Self-sovereign identity is not a new concept. In the physical world, it became the essential concept for identity management. E.g. people own an identity document in the form of a passport, have a membership card of the club and hold a bank card to withdraw money from a cash-point. Such documents are usually kept in a personal wallet or are preserved by the owner in a save place. It is natural for us that the identity is managed and controlled by our own since people started to get used to it around 450 BC when Persian King Artaxerxes issued the first passport [5]. The self-sovereign identity model is about enabling the same concept in the digital world. It is about giving back control of the digital identity to the user and about enabling system interoperability. Alternatively, as Christopher Allen, Co-author of the TLS security standard said:

"Self-sovereign identity is the next step beyond user-centric identity, and that means it begins at the same place: the user must be central to the administration of identity. That requires not just the interoperability of a user's identity across multiple locations, with the

user's consent, but also true user control of that digital identity, creating user autonomy." [13]

He points out that, SSI extends the user-centric model with the possibility of exchanging the identity between domain on consensus with the user. The user should not only benefit from increased usability by a credential management device; instead, he should obtain full control about his identities.

The Figure 1.7 illustrates the SSI model. Service providers are organized in isolated domains, and the user owns an identity management device alike to the user-centric model. Furthermore, there is a globally distributed domain, realized as a Distributed Ledger (DL), which provides a consistent view on shared identifiers and public keys. A user can create, modify or delete his identifiers stored in the DL. Moreover, he controls additional claims and can use them to create identities to register on a service. The Figure 1.2 describes the scenario. The service provider 1 acts as an issuer of a claim (e.g., name, date of birth or address) and transmit it to the user, who stores the claim in his management device. When he registers at a new service, he creates a new identity out of the claims and identifiers in his management device. The key difference to the other models is that a user controls the identifiers and can use of claims asserted by a third party.

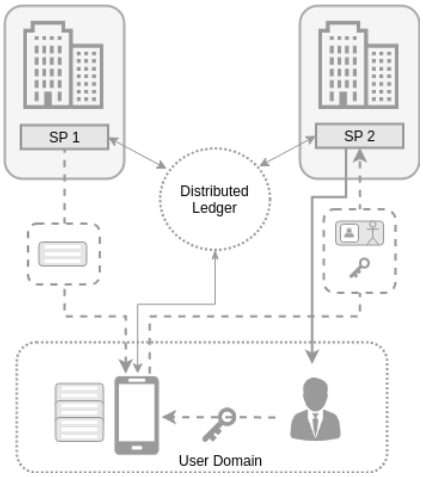

Figure 1.7: Self-sovereign identity: An user controlled identity (Legend: Figure 1.2)

The heart of self-sovereign identity is the user's control over his identity. To ensure full control, Christopher Allen formalized ten principles of SSI [13]:

- **Existence - users must have an independent existence:** A user can never exist wholly in digital form. Behind each SSI there is a entity with distinct existence. SSI makes public and accessible some limited aspects of the entity.
- **Control - users must control their identities:** The user is the ultimate authority on their identities. They should always be able to refer to it, update it, or even hide it.
- **Access - users must have access to their data:** An user must always be able to easily retrieve all the claims and other data within his identity.
- **Transparency - systems and algorithms must be transparent:** The systems used to administer and operate a network of identities must be open, both in how they function and in how they are managed and updated. Algorithms should be free and open-source.

- **Persistence - identities must be long-lived:** Preferably, identities should last forever, or at least for as long as the user wishes. Though private keys might need to be rotated and data might need to be changed, the identity remains.
- **Portability - information and services about identity must be transportable:** Identities must not be held by a singular third-party entity, even it is trusted. The problem is that an entity may disappear, and thus, identities have to be transportable on the wish of the user.
- **Interoperability - identities should be as widely usable as possible:** They are of little value if they only work in limited areas. A digital identity system should make identity information widely available.
- **Consent - users must agree to the use of their identity:** The sharing of identity and claims must only occur with the consent of the user.
- **Minimization - disclosure of claims must be minimized:** When data is disclosed, that disclosure should involve the minimum amount of data necessary to accomplish the task at hand. E.g. to disclose that the identity is over 18 years old, should not be done by revealing its actual birthday. Only the necessary information should be provided. In this case, a proof or a claim that the identity is over 18 years old.
- **Protection - the rights of users must be protected:** When there is a conflict between the needs of the identity network and the rights of individual users, then the rights and freedoms of the individual user should be preserved over the needs of the network. Identity authentication must be censorship-resistant and force-resilient, which requires a decentralized system.

1.3 R3 Corda

Corda [15] is an open-source distributed ledger platform founded by a consortium of international banks. It was developed to steer the data flow to parties only involved in business processes. In contrast to own independent and inconsistent systems, Corda enables interoperability without restriction in privacy. It is achieved by creating a digital model close to the real world's business logic. Parties participating in an agreement share an object only visible to them. Thus, this shared object is not known by parties outside of the group of participants. Corda use the UTXO (Unspent Transaction Output) computational model to record the global state, thus, the latest state of the shared object is an immutable state. This state can be consumed by a transaction to update the shared object by creating a new state. This process of consuming and creating states leads to concatenate of all previous states, or in other words, to a hash chain which is only present between the group. In contrast to other UTXO system like Bitcoin, Corda uses a generalized approach of the concept by supporting arbitrary data models. In this sense, Corda is a distributed ledger with significant differences to traditional

blockchains. In contrast to permissionless blockchains, Corda enables real-world transactions between identifiable parties, with privacy and legal certainty. Moreover, in contrast to permissioned blockchains, Corda allows the co-existence and the interoperation of multiple groups of participants along the same network [15].

Corda key features [45]:

- **Privacy** - Only the parties involved in the agreement have access to a shared object and its transactions.
- **Scalability** - Since data is only shared and stored along the participants of a group (not globally) transactions per time of the network can reach a large scale.
- **Finality** - At the time of completion of a transaction, all parties have an assurance that the transaction is final and cannot be reversed. Unlike in mining mechanism of traditional blockchains[1], where the is a chance that after successfully mining the block get removed by another different block.
- **Identifiable Participants** - Participants of a shared object are clearly identified.
- **Interoperability** - Data shared between some parties can be transferred to any other party without conversion.

1.3.1 Corda in a nutshell

A brief introduction to the Corda peer-to-peer network model and terminology is given in this Subsection [47]. Starting with the terminology:

- **State**: An immutable representation of data.
- **Shared object**: A chain of states linked together with hashes. All states are considered to spend except the latest one.
- **Transaction**: A record which takes null or several states as its input and outputs null or several new states. The correctness of the transaction is verified by smart contracts.
- **Smart contract**: A function set related to a shared object. It defines rules to mutate the shared object via a transaction.
- **Object participants**: Sharing a shared object. They may have to agree on a shared object mutation depending on the related smart contract.
- **Flow**: A full process from creation to the signature collection until the completion of a transaction. It contains the communication to all participants and the notary.

[1] proof of work consensus algorithm as presented in the Bitcoin white paper

- **Notary**: A logical unit, which can be distributed among multiple entities by a byzantine fault tolerance consensus algorithm [32]. It verifies the correctness of a transaction and controls against double spending.

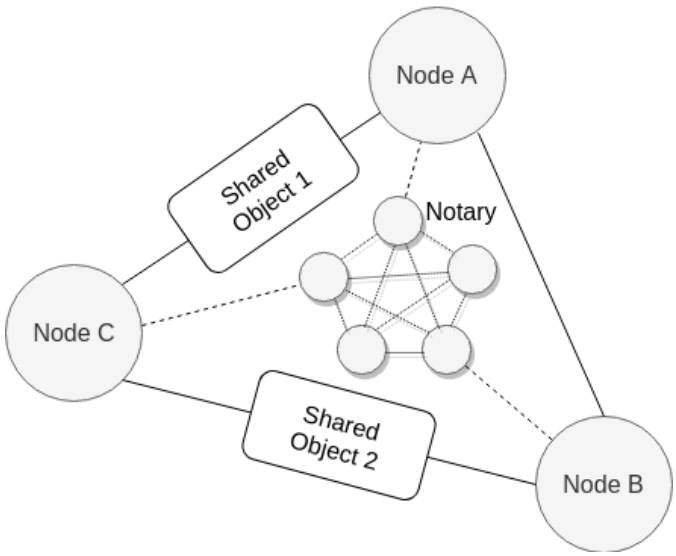

Figure 1.8: Corda network model

Corda network model, Figure 1.8: The network is a full-mesh - all nodes are connected, and thus, can directly communicate with each other. Nodes can agree on shared objects, meaning agree on the data format and smart contract, to have the same view on the data. Nodes can participate in as many shared object as they like. Further, a certain shared object can be shared between multiple nodes. How consensus is achieved to updated a shared object is individually defined in a smart contract. However, each shared object has a notary, which might be a byzantine fault tolerance network [32]. The notary validates the transaction of the specific shared object and checks for double spending.

1.4 Cardossier

The vision of the Cardossier project [43] is to innovate the car ecosystem by creating a data exchange of trusted data via a distributed ledger. Initially, the project started with the question of how to establish more trust in trading used cars [22]. Since the information about the car's history is limited when buying a used car nowadays, we have to trust that the provided information by the seller is correct. The true condition of the car can only be verified superficially by, e.g. taking a look at the engine, rudimentary checking for rust and making a test drive. However, the correctness of some information cannot be verified such as the frequency of maintaining the car, the correctness of the mileage or the existence of hidden

damages caused by an accident. On a first view, it seems less of a big issue, but with taking it into proportion of numbers, the full complexity becomes visible. Each year about 300'000 new cars get a car registration and about 856'000 cars are trade on the used car market only in Switzerland [37], [42]. Most trades are performed on pen and paper what is prone to error. A car history, which is verifiable and transparent, would reveal the real condition of a car. These two desired properties are achieved in Cardossier by making use of Corda. Reaching the goal of a verifiable car history requires to involve all car ecosystem related players starting by the manufacturer to the dealer, insurance companies, registration authorities and private buyers and sellers until to the scrap merchant into the system. All participants provide car relevant data to the network and sign them with their identities to create trustworthy information. Further, a buyer can verify that the seller revealed all data. Otherwise, the seller could hide, for instance, information about car damages. It requires a mechanism to ensure transparency.
Furthermore, added value is created by having connected all parties of the car ecosystem on one single digital system [26]. A secure data exchange can follow without a central authority. Additionally, a standard data format is established, and thus, unnecessary data conversion between the parties can be avoided.
However, the main benefit for companies is the opportunity of business process automation. For instance, an insurance company may offer to enter car insurance in a few clicks on Cardossier and provide the certificate of insurance in a verifiable data format such that car insurance can be proved to the registration authority while asking for vehicle registration. These and similar processes can be digitalized and become more cost and time-efficient.
Car owners benefit from a reduced administrative effort in managing his car by the digitalized processes and data, which is accessible via a single dashboard. Especially, car fleet management may gain inefficiency significantly.

The idea of a verifiable vehicle history report is not new. The platform CARFAX [44] based on a traditional system[2] is already providing similar functionalities. However, it does not offer to the companies to create business processes and to execute them on the platform. Furthermore, their vehicle report is about trusting on a single company which has to collect the data of the full ecosystem. In Cardossier the trust and the data are distributed by the unique architecture of Corda. Each party (dealer, an insurance company or merchandise) run a Corda node which has an identity well-known inside the network. The parties can supply new signed data (e.g. car mileage) to the network by attaching it to the related car object. A smart contract verifies the correctness of the data and controls the authorization. However, the data is not broadcasted along with the system as it is the case in blockchains. The data is only provided to the nodes which are participating in the specific car object. Thus, car-related data is never shared globally. Moreover, trust in data correctness is distributed along with each data provider and the parties participating in transaction validation.

At this point, the private customers have not been brought in yet. They are crucial since they should benefit from a verifiable vehicle report and the automated processes.

[2] A traditional system is referred as a centralized or distributed system which is controlled by a single entity (individual, company or consortium)

1.5 Problem Statement

The previous section provided a general introduction into the field, whereas the detailed problem statement of the thesis is formalized in the first part of this section. In the second part, the scope and delimitation of this work are described.

The general problem definition of this research objective is formalized as:

How can users take control of their identity and interact with the Cardossier ecosystem with privacy and trust?

This general question is further divided into sub-questions; each of them demands a solution for a key problem of the general problem statement. Answering a sub-problem contributes to the general research question.

1. What are the requirements on identity management in Cardossier and can SSI fulfill them?
   Identity management (IdM) solutions are rarely standalone. Usually, the IdM is embedded into a bigger system or ecosystem, and each identity model has different flows and functionalities as presented in Section 1.2. The system demand on an IdM and the desired identity model have to be investigated to provide a frictionless co-existence. In our case, Cardossier and SSI principles which let users control their identities.

2. What are existing data models and implementations of SSI?
   Efforts in realizing self-sovereign identities have been made in recent years. Specifications and proposals for SSI have been presented and implemented. The current specifications, tools and libraries of SSI projects, as well as their proposed future work, should be investigated and analyzed on their re-usability in Cardossier.

3. How can SSI be enabled in Cardossier?
   It is equivalent to the question of how to enable SSI in Corda since Cardossier is a Corda based system. Therefore, it implicitly requests for a solution of how to provide SSI in the permissioned distributed ledger Corda. A concept of an SSI solution should be worked up, which considers the requirements and the pre-known problems:

   - How can the disclosure of personal data be minimized?
   - How can identity correlation attacks be prevented?

4. How can a user give consent to Corda transactions?
   At this point, users cannot authorize Corda transactions. It is a desired feature for claiming vehicle record ownership and giving consent to an ownership change. Since SSI may provide a solution to enable the functionality, a concept should be worked up to realizing it.

5. How can SSI be utilized in Cardossier?
   SSI provides base functionalities which have to be embedded in Cardossier specific

processes. Answer to the question of how a general process flow will look like should be given.

6. What is a suitable architecture to enable the solutions?
   A software architecture to realize the solution should be presented.

After stating out the research question and structuring it into sub-problems, we define the scope and delimitation to allow a clear work definition.

1. A solution proposal about the architecture and concepts will be presented. Partial implementations on the scope of a proof-of-work should be made.

2. It will be discussed how identity correlation can be avoided under the umbrella of IdM solution. Correlation via external sources such as internet traffic will not be considered.

CHAPTER 2

Research

In this chapter, we discuss emerging specifications and published concepts for SSI in Section 2.1. Furthermore, we describe the most popular projects which provide self-sovereign identities, in Section 2.2.

2.1 SSI: Draft Specifications and Related Concepts

In this section, the fundamentals of SSI are discussed. The crucial element of each SSI system is a decentralized public key infrastructure (DPKI), which have been conceptually described by the Rebooting the Web of Trust organization. In a first step, the concept of DPKI is discussed (Section 2.1.1) before investigating the emerging W3C standard of decentralized identifier (DID) in a next step (Section 2.1.2).
DIDs can be enabled by a various systems with different properties, consequently the systems will operate differently. Thus, the Decentralized Identity Foundation (DIF) has presented a universal resolver which enables interoperability between DID providing systems, and therefore, DIDs become globally usable. The universal resolver is examined in Section 2.1.3.
A W3C Community Group has presented a draft standard for verifiable credentials. This standard describes the data model and approaches to create third-party verifiable credentials based on DID's as well as zero-knowledge capable credentials, see Section 2.1.4.
Data containing personal information should always be controlled by the identity owner as well as the data stored on the cloud. To ensure data control, DIF presented the concept of an identity hub to enable a remote data storage which utilizes DID's as identity. The identity hub is described in Section 2.1.5. Finally, in the last section, a different concept of a PKI is discussed, the FIDO2 standard for user-centric identities (Section 2.1.7).

2.1.1 Decentralized PKI

Communications and interactions in the digital domain are secured through asymmetric encryption, which requires a safe exchange mechanism of public keys, e.g. like described in the TLS standard [6]. Sender and receiver exchange their public keys, encrypt their messages with the counterpart's public key and decrypt the messages with its private key. However,

there is a missing element. How do the parties know that they communicate with the right counterpart? The public key infrastructure (PKI) enables trust and proof of identity by involving the use of certificates and trusted third parties [2]. Certificate Authorities (CA) issue a certificate that allows validating the integrity and ownership of the public keys. Worldwide, there exists only a few large CA's which provide the internet with certificates.
The concept of decentralized PKI (DPKI) has been proposed by the organization Rebooting the Web of Trust to resolve the issues of the traditional PKI [12]. They outlined several problems. Data might be managed in one company's repository which leads to a single point of failure. E.g. a web hosting company, which is responsible for the key management of its clients and stores the keys in their repository, creates a potential security risk. In case the repository is compromised the security of the clients' websites may be broken. Another outlined problem is fraudulent certificates caused through the limited number of CA's. If the authority were compromised, it would allow seamless Man-in-the-Middle attack on each certificate the authority has issued. The decentralized PKI (DPKI) approach addresses these problems by diffusing the trust around all participating entities, which avoids the need of centralized authorities or key centralization, and thus, no malicious party can compromise the integrity of the system. DPKI focuses primarily on distributed ledgers which provide decentralized key-value data storage. They propose direct control and ownership of a globally readable identifier for the identity owner by registering identifier and public keys at a blockchain. Each participant has the same view on the blockchain and can link an identifier's lookup value to the latest public keys [12].

2.1.2 Decentralized Identifier (DID)

Decentralized identifiers are, as the name suggested, unique identifiers based on a decentralized PKI. They are defined in a draft specification by W3C [33], and they are based on the fundamental concept to enable SSI.
In general, a DID consists of DID tag, DID method and an id-string (e.g. did:example:123456789abcdefghi), whereas the tag points out that this is a DID and the method refers to the environment of the DID. It is not intended that the id-string is human-meaningful, but a random string of characters. Otherwise, the capability of generating globally unique identifiers is automatically not given as the Zooko's Triangle: "human-meaningful, decentralized, secure — pick any two", demonstrates [33]. Each DID belongs to a DID document which contains meta-information such as public keys and service endpoints. The DID document, which is created, read, updated and deactivated (CRUD operations) through the DPKI, makes the concept compelling. It is fully managed by the DID owner (or empowered delegate). It is publicly accessible on a distributed ledger to guarantee each participant the same view on the document and to make it unforgeable. Important to mention is that the DID document does not contain any personal information about the owner itself.

Listing 4.2.9 shows a simple example of a DID document. Each document is written in JSON-LD (see Section 2.1.6) data format and contains the necessary element "@context", which points to the location of the data schema. This principle of linked data structure should

allow for better development of machine-to-machine communication. Further explanation about JSON-LD can be found in Section 2.1.6. A "@context" value can be specified for each DID method, but it must contain the general DID context. Every DID with the same DID method must have the same context value. The next property, the "id" specifies the DID to which this document belongs to. Further, one or more public keys of the DID owner are provided by the "publicKey" property. It is very similar to commonly known internet certificates. The "authentication" property defines keys to use for authentication. Each DID must have at least one public-private cryptographic key pair for authentication. Thus, the owner can prove ownership through a challenge-response cryptographic algorithm. In order to ensure trust and reliability, it is crucial that the private keys are protected with a strong security mechanism by the owner. Last but not least, services which the owner offers can be defined in the "service" property.

Listing 2.1: An example of a DID Document [33]

```
{
  "@context": "https://w3id.org/did/v1",
  "id": "did:example:123456789abcdefghi",

  "publicKey": [{
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "RsaVerificationKey2018",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyPem": "-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n"
  }],

  "authentication": [
    "did:example:123456789abcdefghi#keys-1"
  ],

  "service": [{
    "type": "OpenIdConnectVersion1.0Service",
    "serviceEndpoint": "https://openid.example.com/"
  }]
}
```

DID Method Specification

The DID concept is designed to create decentralized identifiers on distributed ledger or network, but it is not limited to that. The centralized system may add support for DID's to create an interoperability bridge between centralized, federated and decentralized identifiers. DID document operations such as create, read, update and deactivate (CRUD operations) have to be done differently on each distributed ledger or other systems. Therefore, each DID belongs to a namespace with specific CRUD operations which are defined in the DID method specification. At a minimum the following attributes have to be specified to register a new DID

method at the W3C community:

- DID method name
- Target system
- Namespace specific identifier including DID generation
- Context definition
- CRUD operations
- Security and privacy considerations

Since each DID method has a different underlying system, and thus, another mechanism to perform the CRUD operations, each DID is isolated in its namespace. Hence, a universal resolver (see Section 2.1.3) has been introduced to break the isolation and to reach interoperable DID's.

2.1.3 Universal Resolver

A universal resolver has been introduced by the Decentralized Identity Foundation (DIF) [19] to make DID's globally resolvable. The tool is similar to the DNS system which resolves an identifier in the form of a domain name to a website or web service. Figure 2.1 illustrates the concept. The resolver contains the drivers for the different networks which are selected depending on the DID method. A driver is the implementation of the CRUD operations for the corresponding network. As a result, DID's can be used globally through a single resolution entry, and thus, an application has not to care about specify network drivers.

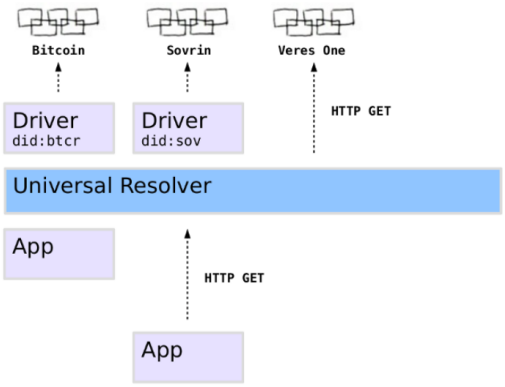

Figure 2.1: DIF universal resolver [19]

2.1.4 Verifiable Credentials

Credentials are an essential part in our daily lives: identity card, passport, driving licenses, credit card, employee badge, etc. are used to assert that we have citizenship, we are capable of driving a vehicle and we have access to accounts or buildings. However, they are mainly limited to the physical world and are not present in the digital world except for login credentials. The W3C Verifiable Credentials Data Model [35] specifies an ecosystem which enables cryptographically secure, privacy-respecting and machine-verifiable digital credentials,

which can be used to replace physical credentials to establish trust at a distance and to automate processes, and thus, to become more efficient. Moreover, verifiable credentials provide a basic building block to enhance privacy on the web. Personal information like healthcare data of financial account details can be managed in an owner-controlled manner.

Credentials can have different meaning depending on the discipline. However, a general definition can be given as attested information about qualification, competence, or authority to an individual by an issuer [28]. Commonly, credentials can be verified by any third party and the verification process requires no interactions with the issuer, and thus, it is privacy-preserving. These two key attributes are enabled in W3C's verifiable credentials by making use of decentralized identifiers (DID's) on its base. Verifiable credentials can be perceived as an extension of the DID concept to claim additional information on a DID or an entity. However, it is not limited to it, and any other sufficient identifier might be used.

The specification describes four roles: an issuer, a holder, a verifier and a verifiable data registry whereby the last one is a publicly accessible registry such as trusted databases, decentralized databases, government databases, and distributed ledger [35]. On this registry, identifiers and credential schemas are stored. An issuer may register new schemas or use a schema from the registry to issue a verifiable credential to the holder. The holder owns an identifier which is registered in the registry as well and is used to authenticate against the issuer. After issuance, the holder stores the verifiable credential inside its vault. In case he is required to use the information of his verifiable credentials, he creates a verifiable presentation which ideally contains only the minimum required information. Each verifiable credential contains proof of the issuer. These proofs are a part of the verifiable presentation, and they allow to verify the authenticity of the provided data of the presentation. The verifier checks the schemas, the identifiers and the proofs of the presentation without interaction with the issuer since all needed information is accessible from the registry. However, it requires knowledge and registration of the identity of the issuer.

Verifiable Credential Data Model

A verifiable credential consists of three components: the credential metadata, the claim(s) and the proof(s). Metadata describe the properties of the credential such as expiration date, the issuer, the type of credentials, or credential identifier [35]. A claim is an assertion which the issuer makes about the credential subject. It is a subject-property-value relationship as illustrated in Figure 2.2, and thus, each claim contains single information such as the name, the birthday, etc. about the subject. Finally, a verifiable credential is certified by one or more digital signatures of the issuer.

Listing 2.2 shows an example of a verifiable credential. As in the DID concept, the data format JSON-Ld is used, and thus, "@context" property is provided at the beginning of each credential followed by the metadata (id, type, issuer, etc) and a single claim (credentialSubject). In the end, a digital signature is provided to protect the integrity of the document and to allow verification.

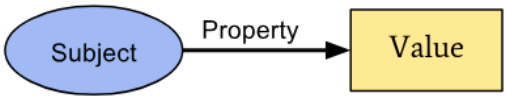

Figure 2.2: A claim, a subject-property-value relationship [35]

Listing 2.2: An example of a verifiable credential [35]

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://www.w3.org/2018/credentials/examples/v1"
  ],
  "id": "http://example.edu/credentials/1872",
  "type": ["VerifiableCredential", "AlumniCredential"],
  "issuer": "https://example.edu/issuers/565049",
  "issuanceDate": "2010-01-01T19:73:24Z",
  "credentialSubject": {
    "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
    "date of birth": "01.01.2000"
  },
  "proof": {
    "type": "RsaSignature2018",
    "created": "2017-06-18T21:19:10Z",
    "creator": "https://example.edu/issuers/keys/1",
    "jws": "eyJhbGciOiJSUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19
  }
}
```

Verifiable Presentation Data Model

The model is similar to verifiable credentials. Instead of having claims providing the information, verifiable credentials are listed next to metadata and proof(s) [35]. A presentation can then be validated by verifying each credential proof. In general, a proof might be simply a digital signature. However, the model even supports more advanced proof schemas such as zero-knowledge proofs (ZKP's), see Section .

Zero-Knowledge Proof of Verifiable Credential

A zero-knowledge proof is a cryptographic method which allows proving the knowledge of a value without disclosing the actual value. For instance, we know a password for an account, and we prove that we know the password without leaking to a third party. From this basis, more sophisticated ZKP's have been introducing, such as proving your degree from a univer-

sity without revealing your identity. In the context of verifiable credentials, the following key capabilities can be achieved through ZKP mechanisms [35]:

1. A credential holder can combine multiple verifiable credentials from multiple issuers into a single presentation without revealing credential or subject identifiers to the verifier. It is a critical capability when it comes to preventing deanonymizing through correlation.

2. A credential holder can selectively disclose the claims of a verifiable credential. It allows providing only necessary information to a verifier.

3. A credential holder can produce a derived credential that is formatted according to the verifier's data schema without needing to involve the issuer. This increases the flexibility of a holder to use its credentials.

Noted that point two could be achieved by issuing multiple atomic credentials (a credential which contains only one claim). However, it would result in credential dependencies and increase the complexity in credential management.
ZKP of verifiable credentials requires an additional extension to the data model. A verifiable credential needs a proof which supports ZK, and thus, each credential has to be issued in ZK manner to be used in a presentation under ZK. Furthermore, a credential must contain a credential definition to perform zero-knowledge operations. This credential definition has to be accessible by the verifier, and thus, should be stored in the verifiable data registry.
The ZK property increases the benefits of verifiable credential drastically. It enables great flexibility for the holder and strong privacy through minimum disclosure of personal information - a property which is not feasible with physical credentials.

2.1.5 Identity Hub

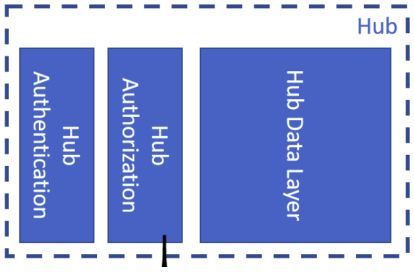

Figure 2.3: DIF identity hub [36]

The Decentralized Identity Foundation has proposed the concept of the identity hub (DIF) [36]. At its core, an identity hub is a cloud data storage (like Dropbox, google drive, etc.), and therefore, it provides the basic functionalities of uploading, downloading and sharing data. Since it is DID based, it differs in the identity management. A hub instance belongs to a DID, and thus, the ownership relies on a public/private key pair of the DID. A hub consists of three layers, see Figure 2.3, an authentication layer, an authorization layer and the data layer: The authentication layer is a challenge-response authentication schema [36], read-write permissions for each data object are managed fine-grain by the authorization layer, and the data layer manages the data storage.
The owner controls the permissions and the data, whereas a hub service provider might

provide the hub instance. All data might get lost when the hub service provider becomes bankrupt, or he decides to refuse access to the owner. An issue which is addressed by providing a steamless synchronization mechanism between multiple hub instances. It allows a user to own several auto-synchronized hubs at various providers. In the case a provider refuse the access, the data will still be accessible on another hub instance. The identity hub API is an open standard and the DIF has made an implementation publicly available under free software licences [49]. It should enable that a DID owner can run its identity hub, and thus, become fully independent.

2.1.6 JSON Linked Data (JSON-LD)

JSON-LD is a W3C syntax standard for linked data based on JSON [11]. It is used to create machine-interpretable data across different documents and websites, which allows an application to start at one piece of Linked-Data and following embedded links to other JSON-LD documents that are hosted on different websites. The motivation is to provide a standard to build interoperable web services with slightly changes on the widespread JSON standard.

2.1.7 The Web Authentication API

The web authentication API is a user-centric identity management solution for user authentication based on a key pair [46]. It was presented by the FIDO Alliance to reduce the reliance on passwords to authenticate users. Nowadays, all main web browser support the API. Figure 2.4 shows involved elements. A user posses an authenticator - any hardware which is able to store a private key and communicate with a user agent. Currently, the smartphone and USB Token/Yubikey are the most prominent FIDO authenticatiors. When the user is login to a web page via the browser, he get asks from the relaying party (server) to sign a challenge. The user connects his registered authenticator with the user agent, which forwards the challenge and receives back a signed authentication token. Then the browser forwards the token to the relying party [38], [53].

2.2 Related Projects

Various projects are focusing on the concept of SSI. In this section, the three most dominant projects/concepts in the area of SSI are discussed.
The Hyperledger Indy project is the leading project regarding SSI. It has the strongest community in which several international companies participate. Furthermore, the developed system is currently the only one with zero-knowledge properties. It supports to proof credentials and ownership under zero-knowledge. However, it is also the only project which has developed its permissioned blockchain, only for SSI. So far, no full-fledged application has been provided since it is focused on developing tools and libraries, whereas the implementation of an application is left over to commercial providers.

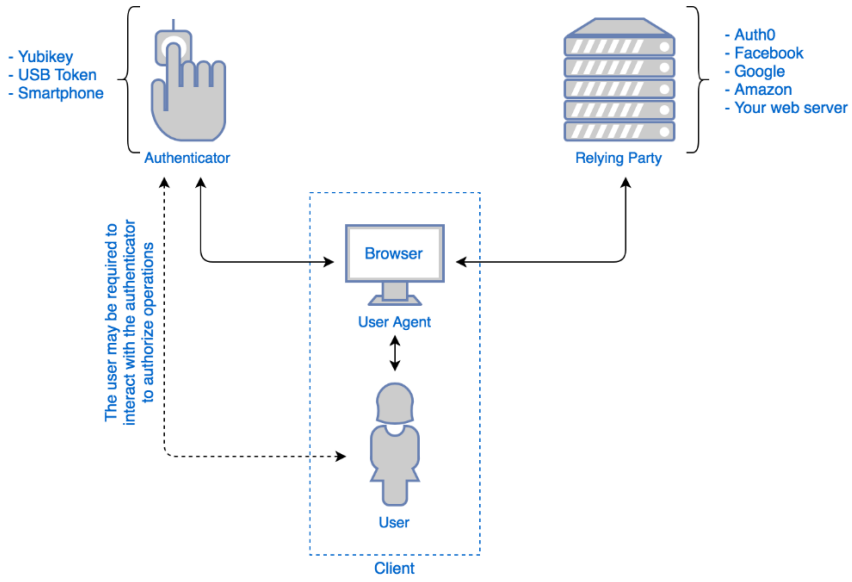

Figure 2.4: Web authentication API [53]

uPort make use of the permissionless blockchain Ethereum as its ground source, and thus, it enables to use the identities also by Ethereum transactions. uPort provides a complete open-source system from its own DPKI to tools for verifiable credential and a mobile app. Moreover, uPort is already used in production. For instance, citizens of the city of Zug, Switzerland can get their digital identity card in the form of a verifiable credential on the uPort mobile app [25].
BlockCerts allows creating digital academic records or degree certificates in the form of a verifiable credential. The system is in production as well, but it is strongly limited and less featured. However, BlockCerts is discussed since it makes use of a different approach to utilize the blockchain for SSI
At this point, it has to be mentioned that several additional projects are working on SSI. However, most are less developed or using a similar approach as the three projects discussed mention above. For instance, Jolocom, a start-up headquartered in Berlin, Germany, has little difference to the implementations and features of uPort [20]. Furthermore, there are various projects utilizing a blockchain in the same way as BlockCerts does, e.g. TalentChain, ShoCard, OpenBadges.

Since there is also the giant corporate Microsoft contributing to SSI, we briefly discuss its activities in the last part.

2.2.1 Hyperledger Indy

Hyperledger Indy is an open-source project hosted by the Linux Foundation. The aim is to provide tools, libraries and reusable components for creating decentralized digital identities [48]. All its project focus is set to develop the fundamental building blocks to create self-sovereign identity ecosystems. There are two code bases on its centre of development, the indy-node and indy-sdk, whereas the node as the name suggested is an implementation of a blockchain node and the SDK a tool kit for creating and using SSI. The node is designed with scalability and robustness in mind, and therefore, it is an implementation of the Redundant Byzantine Fault Tolerance (RBFT) [9] consensus algorithm. Thus, the indy-node allows creating permissioned blockchain networks. The SDK enables interactions with an indy network (indy-pool). It allows us to put DID documents, schemas, etc. onto the ledger, and resolve them. Furthermore, it contains an implementation of a wallet, which is used to create and manage DID's and verifiable credentials as well as to manage the private keys. Additionally, zero-knowledge capable credentials are supported, from issuance to proving a claim under zero-knowledge. It is an implementation of the Identity Mixer and U-Prove presented by IBM Research and Microsoft, respectively [1], [10], [50]. It enables the following four capabilities [48]:

- **Hiding claims:** Not required claims of a verifiable presentation can be hidden in the verifiable presentation while the validity of all other claims can still be proven.
- **Proof of having a claim:** The claim is not revealed - only the knowledge is proved.
- **" I am/have ..":** It is different to "Alice is/has .." since no identifier is required. Verifiable credentials can be proven without revealing the DID to which they have been issued. Instead, the prover proves the knowledge of a secret which is linked to each claim.
- **Minimum/maximum proof:** It can be proven that a claim is smaller or bigger than a certain threshold.

Besides, Indy has implemented a system to prove non-revocation of credentials. Instead of a revocation registry - a list of credential ID's and revocation state - the identity owner proofs the revocation state to the verifier. Cryptographic accumulators enable it. It can be thought of an accumulator as the product of multiplying many numbers together. For instance, in equation $a * b * c = e$ the accumulator would be e. Let us assume $a = 3, b = 5, c = 7$ then e has a value of 105. Because 3 is a factor of 105, we say that a is "in" e. If 3 should be taken out of the accumulator, 105 is divided by 3 and 35 results. 3 has now been removed [31]. An identity owner can prove to a verifier that he knows a number which is in the accumulator. If he can make such a proof, the credential is valid. Otherwise, it was revoked by the issuer. All credentials of the same type have a single accumulator, and a credential belongs to a certain factor, on which only the issuer and the identity owner know about.

There are several projects making use of Hyperledger Indy to build up their network. The most popular and the one having the most significant impact is Sovrin, see the section be-

low. However, there are others, for instance, the Verifiable Organizations Network from two Canadian states [59].

Sovrin

Sovrin Network is a Hyperledger Indy based network. It is a permissioned public blockchain, which is managed by the non-profit Sovrin Foundation [54]. The foundation leads the open-source community effort (Hyperledger Indy) to develop further and maintain the network. Furthermore, it defines a trust framework to establish SSI in the network, and it is responsible for recruiting new parties who participate in the network via a node, called stewards. Additionally, the foundation aims to guarantee public accessibility.

However, the foundation does not lead the development of the complete application stack [17]. Actual, only the network (DID layer) is declared as a non-profit public utility whereby cloud services (cloud layer) and client-side applications (edge layer) are left over to the competitive market. Figure 2.5 the layer architecture. The DID layer is the global state and the fundamental component of the system. It enables to create DID as described by the W3C standard for decentralized identifiers [33]. The cloud layer, as well as the edge layer, get access to the ledger via a steward - a node of the Sovrin network. The utility of the cloud layer is only optionally required since it is not necessary to enable W3C's verifiable credentials [35], but desirable to make the system more convenient for its users. The cloud layer serves four core functions [17]:

- **Persistent messaging endpoints:** Clients operating on an edge device is typically not directly addressable on the internet, but via a network service running on their behalf like email servers, IP routers or other services. However, the addresses, in this case, are selected by the service provider and cannot be specified by the user. In Sovrin, a user can add, change and remove cloud agent endpoints - addresses to any communications (secure messaging, file sharing, VoIP, etc.).
- **Coordination endpoints for multiple clients:** A client may have multiple edge devices (smartphone, laptop, etc.) with the same identities, and thus, the messages incoming from the endpoints have to be spread along with all its devices.
- **Encrypted backup of Sovrin keyrings:** The keychain is the heart of each identity wallet. Cloud agents may provide backup mechanisms for key recovery.
- **Encrypted data storage and sharing:** Data storage on the cloud as traditional providers offer, but with the add on that the data should be encrypted and managed by the identity owner's keychain.

The cloud agents make the system more user-friendly by enabling messaging, data storage, portability and backups, however, identity owners are always able to communicate with the DID layer directly to be independent of cloud agents for DID operations and validating transactions. The edge layer contains all client-side applications. The most important is the wallet

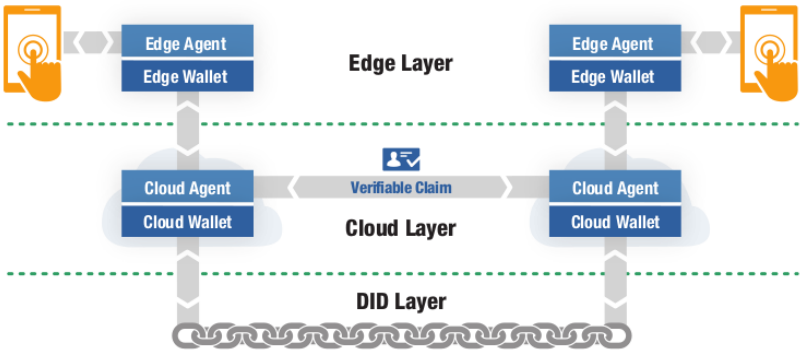

Figure 2.5: The sovrin architecture consists of three layers: DID layer, cloud layer and edge layer [24]

which is used to store and manage the client's private Ed25519 [8] keys (keychain), and also, its verifiable credentials. The basic building block is actively developed by the Indy community, whereas commercial providers should implement full-fledged applications for all kind of devices.
Besides the protection of the keychain, an elementary question to solve is how to recover its keys from key loss or compromise. Sovrin uses a community recovery concept [17]. An identity owner requires to designate their own set of trustees - any other identity owner from a single person to organizations which he trusts to recovery his key if and only if he is asking for it. If this is the case, a certain threshold number of trustees has to sign a transaction to alter the public keys of the DID Document. The threshold can be specified by each identity owner individually. The transaction is validated against the most recent DID document transactions by the validator nodes (stewards) before a specified timelock period. The timelock period is required to be able to recover in case of a key compromise when the attacker has changed the trustees on the DID document. Only when the timelock expired the changes become fully valid, and thus, the identity owner has a time frame to recover.

2.2.2 uPort

The uPort project has started to create self-sovereign identities anchored on the Ethereum blockchain, and to solve the problem of private key recovery [51], a general problem in blockchains. It is a start-up company and highly community-driven with a mindset for free software. The complete project is publicly available under free software licences. They developed a smart contract[1] architecture which allows attaching attributes to an identifier, selectively disclose its attributes and perform social key recovery. During the project lifetime, several adoptions have been made since the initial white paper and first implementation [57], some core changes: the W3C standard for decentralized identifiers (DID) [33] was applied, the external

[1] a smart contract is a program which executed on the Ethereum Virtual Machine

storage of the identity document on the decentralized file system IPFS was replaced with an on-chain storage, and a funding service was added to allow users to interact without owning Ether[2].

Figures 2.6 shows the uPort architecture. It consists of several smart contracts to establish decentralized key management. A user has an identifier in the form of a DID, a private key for creating signatures and a public key in the form of a DID document on the DID registry [58][3]. Optionally, additional public attributes may be stored in the DID document. The identity owner starts with the mobile app, which holds the private keys and allows to manage its identity through interaction with the ledger. Let us assume the identity owner wants to update its public key. He creates the transaction for updating the key, but since he may not have any funds (Ether), he can make use of a funding service which is offered by uPort [55]. Going further, the entry contract for the identity owner is the Controller contract, which maintains access control and recovery logic [51]. The recovery is made via specified recovery delegates, which can be defined by the identity owner. A quorum of delegates is then able to perform a key rotation in the name of the identity owner in case of key loss or compromise. The proxy contract forwards the transaction. Its only purpose is to act as a permanent identifier, which is the contract's address [51]. In our case, we are updating the public key of the DID. The transaction is forwarded to the DID registry which executes the update and stores the new public as an event on the ledger. Once stored on the ledger, an external DID resolver can retrieve the DID document containing the new public key by looking up all events of the identity [52].

At this point, it has to be mentioned that the components of the previously explained architecture have been developed and deployed on the Ethereum testnets and mainnets, but the deployed uPort mobile wallets support not all functionalists on the app stores. The functionality of adding further public keys and service endpoints to its DID is missing. Either social recovery is facilitated, nor key rotation can be done. The only recovery via seed, which has to be stored somehow physically by the user, is supported.

Additionally to the above Ethereum based Decentralized Public Key Infrastructure (DPKI) uPort provides libraries for the https DID method [29]. In this setup, the DID Document is not stored on the ledger, but on a regular webpage under a well-known path. To be able to retrieve the document, it is required to have the domain name as a part of the DID. Hence, the DID has the following format: did:https:example.com. The set up allows maintaining the DID Document at zero cost since no transaction fees occur while providing the same features as hosted on the blockchain. However, it is a wrapper for the centralized PKI to make it compatible with the DPKI since the identifier, which is the domain name, rely on the authorities of the Domain Name System (DNS). Furthermore, the identifier reveals information about the identity which might not be desired.

Besides tools and libraries for a DPKI, uPort has implemented a variant of verifiable credentials based on JSON Web Tokens (JWT's) [56]. It is an extended version of W3C verifiable

[2] Ether is the currency on the Ethereum blockchain

[3] Notice the uPort registry is deprecated and was replaced with the DID Registry

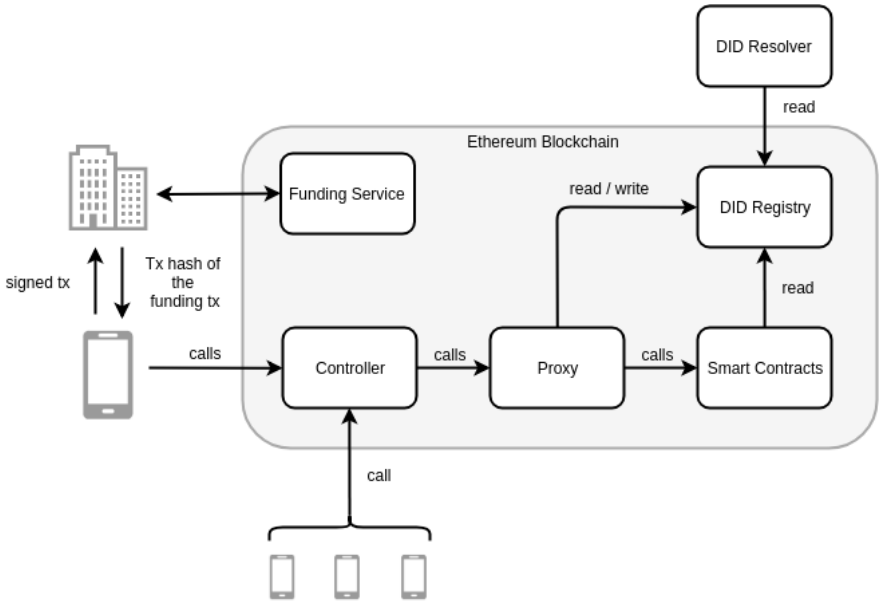

Figure 2.6: uPort architecture

credentials standard [35]. It allows issuing and verifying credentials via an interaction flow initialized with a QR code. The uPort mobile wallet user retrieves a disclosure request by scanning a QR code provided by the verifier/authenticator. The request contains required attributes and a callback. After the user has selected his credentials, which he is willing to disclose to provide the requested attributes, a signed response is created and send back to the verifier.

2.2.3 BlockCerts

The goal of the BlockCerts project is to digitalize academic records and to learning achievements [16]. It started as a research project at the MIT Media Lab, and it has been commercialized by the start-up company Learning Machine. Their vision is tamper-proof digital degree certificates which do not rely on a centralized authority. It has been realized by making use of blockchains properties of being tamper-proof and decentralized. At the time of that writing, BlockCerts provides an issuer and a verifier based on the Bitcoin or Ethereum blockchain as well as a wallet to manage its credentials.

Figure 2.7 illustrates the process of issuance a certificate [21]. First of all, the issuer, e.g. a school has a private key, and a public key which the school publishes on its webpage and the recipient owns a blockchain address (public/private key). The authentication of the recipient has already happened via any possible channel (e.g. in person, credentials from the school etc.). In the first step, the issuer creates and sends the recipient a certificate offer. In return, an address on the blockchain is received. The issuer creates a SHA256 hash of the certificate [16] and he issues it onto the blockchain address. Afterwards, the certificate is transmitted

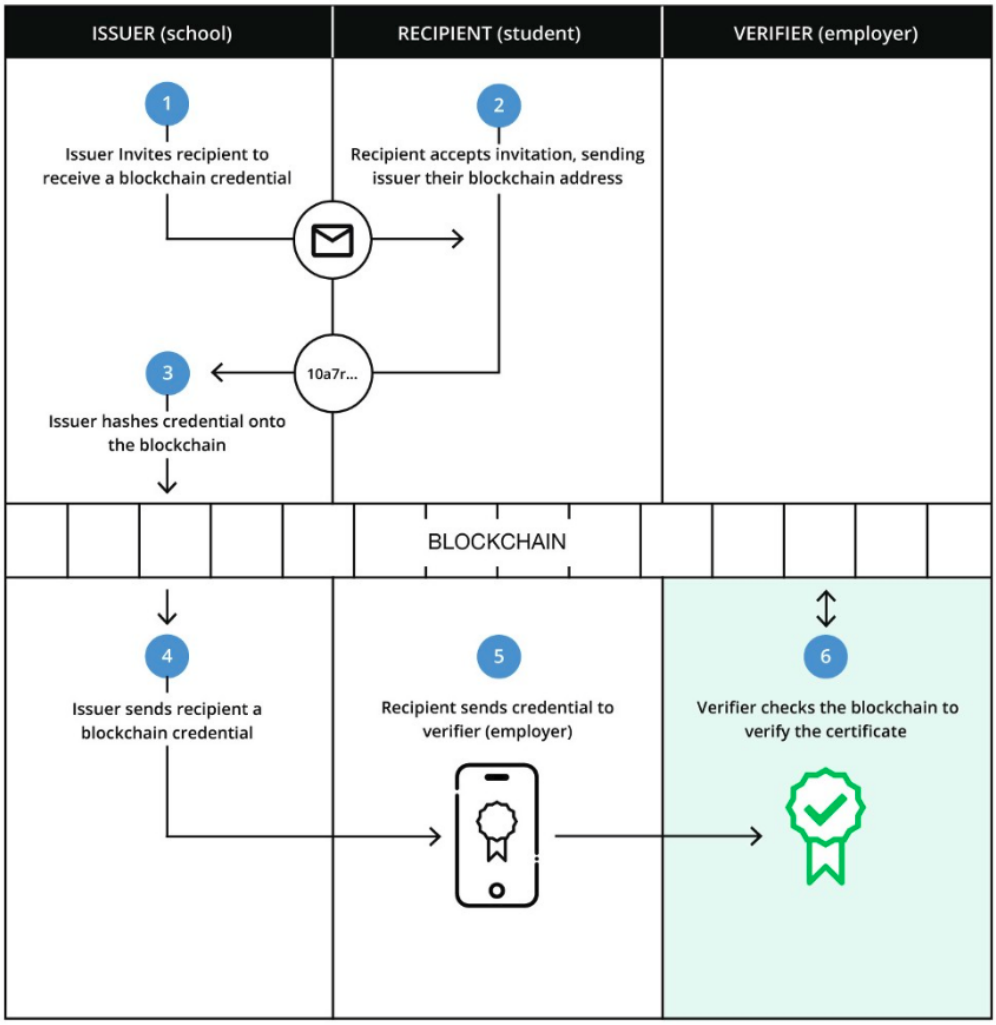

Figure 2.7: BlockCert certificate issuance process [21]

to the recipient's wallet. Finally, a verifier can verify the certificate in five steps [16]: (1) computing the hash of the certificate, (2) fetching the hash from the blockchain, (3) comparing the hashes, (4) getting the public key form issuer's webpage and checking the signature, (5) checking not revoked by issuer.

The system is limited to the specific use case of creating verifiable certificates as described above. Moreover, the W3C standard of decentralized identifiers (DID's) is currently not supported.

2.2.4 Microsoft

Microsoft has proposed the ecosystem of decentralized identities in a white paper [23]. Only recently, they announced a Identity Overlay Network (ION) which runs atop of the Bitcoin blockchain [27]. The system used the IPFS file system to store DID documents and related operations. Similar to uPort, consistence is ensured by anchoring the data via a hash on the blockchain. It is a system based on the sidetree protocol developed by the Decentralized Identity Foundation [34]. Further, they contribute to standard definitions, high-level API's and data storage. They participates in the W3C Credential Community Group which specified the W3C standard of decentralized identifiers [33] and the W3C standard of verifiable credentials [35]. Furthermore, they participate in the specification and development of the DIF DID resolver 2.1.3 and identity hub 2.1.5.
Their main focus lies on high-level API's intending to support the interoperability of DID's between the various system implementations such as Hyperledger Indy and uPort. At the time of that writing, they have to define API's for DID creation, DID resolution, DID authentication and identity storage [41].

CHAPTER 3

Analysis

In this chapter, we discuss the requirements for a identity management system in Cardossier. Beforehand, we categorize the occurred data in Cardossier into three data types in the first section 3.1. In the section 3.2, we present the system requirements. Finally, in section 3.3, we evaluate Hyperledger Indy, uPort and BlockCerts based on our requirements.

3.1 Cardossier Data Types

The Cardossier domain has three data types, see Figure 3.1:

- car/utilization related data: all data which belongs to a car such as import data, registration, mileage, crash, owner, insurance policy, etc.
- personal data: all data belonging to a person or legal entity such as name, address, passwords, age, account, identification number, car insurance record, insurance policy, etc.
- business data: all data involved in business processes such as insurance models, offers, personalized tariffs, etc.

whereby there is an overlap between the types. Some data is part of two types like insurance police or personalized tariffs.
Each data type is differently handled, so it fits privacy, availability and trust requirements best. Car/utilization related data is managed via events and always linked to a car object in the Cardossier network. The personal data should be managed in a privacy conserving manner. Verifiable credentials/certificates should be used such that data can be verified by a third-party. Finally, the business data are managed by the individual organizations itself on their favoured systems.

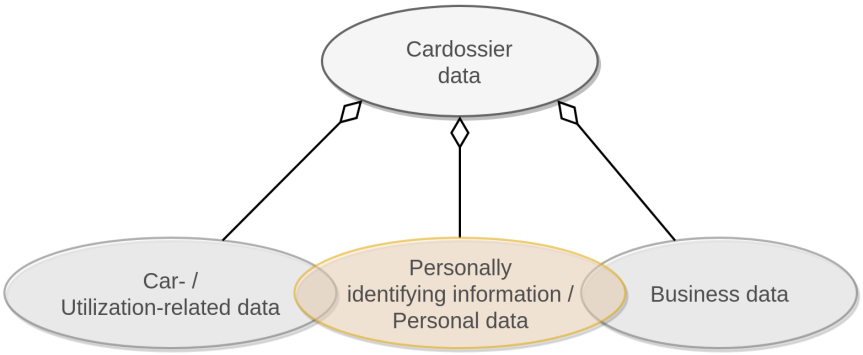

Figure 3.1: Cardossier data types

3.2 Identity Management System Requirements

Identity management system requirements have been worked out by analyzing several use cases in Cardossier. However, we do not discuss them in greater detail since it would go beyond the scope of this document. Instead, we demonstrate why self-sovereign identities should be used in Cardossier by discussing an example of a general use case.

There are two ways of how an end-user can interact with the Cardossier ecosystem: (1) the user run its node or (2) he accesses the ecosystem via a node run by an organization. The focus of this work is set on the second approach.
On a first view, an isolated model, where the user registration at any party providing access to the network might be sufficient. Such a party is called agent throughout the rest of this report. However, an isolated model comes to a limit when interactions between the user and any party of the network are desired. For instance, a user wants car insurance form his preferred insurance company. In this case, the information about the car (vehicle record) is required, including the identity of the car owner. There are two options to enable such an interaction:

1. the agent (Cardossier node operator) is authorized to forward the user's identity and the vehicle record to the insurance company; having federated identities.

2. the user provides his identity directly to the insurance company via an additional authentication process, and he authorizes the agent to share the vehicle record with the insurance company.

In both cases, the insurance company benefits by the verifiable vehicle record, which may allow offering a customized insurance policy.
Option 1 asks for a federated model since the agent forwards the identity to another entity. A drawback is that the agent may require a more detailed digital identity from the user which fulfills the requirements of the insurance industry. Also, the agent must ensure that the identity data is valid. Furthermore, it has the consequence that the issued insurance certificate is

stored at the agent such that it can become part of a new identity. In conclusion, it has to be completely trusted on the agent, which might be a reason for some user not to participate in Cardossier.
In option 2, the agent acts only as a network entry for the vehicle record; no personal information has to be shared. However, how can the user provide an identity to the insurance company without any trusted third-party like in option 1? It requires that the user can manage its identities on a private device. It should have a high usability, otherwise, users may not use it or misspend it. Moreover, the identity data have to have a verifiable format such that the validity can be verified by any third party. These problems are addressed by self-sovereign identity (SSI).

The example above describes only a single use case for identities in Cardossier. However, it already shows certain requirements on identity management (IdM) system in Cardossier such as (3) privacy-preserving identities, (6) verifiable credentials/certificates and (7) user-owned identities and credentials. Additional requirements have been worked out from other cases. It leads to the complete list of requirements in Table 3.1 and Table 3.2. Essential to mention at this point, a privacy-aware system is the central aim of the Cardossier project in general. It is essential to gain confidence in Cardossier of the complete car ecosystem which includes the users as well as the companies. Therefore, privacy affects strongly on the IdM system requirements.

IdM system requirements: Table 3.2 and Table 3.2

3.3 Evaluation of existing Projects

In this section, we evaluate the three SSI projects, Hyperledger Indy, uPort and BlockCert based on our requirements. Finally, we answer the question whether one of them can be used to realize SSI in Cardossier?

3.3.1 Hyperledger Indy / Sovrin

As mentioned above, Sovrin is a Hyperledger Indy based public network. Therefore, the two projects are discussed together. Table 3.3 illustrates how individual requirements can be fulfilled by using Indy. The tool and library provide almost all desired functionalities except (R10) issue credentials as a group member is not supported. (R1) The SDK support to create multiple DID's and to store their key pairs in a wallet, which also provides the (R2) functionality for authentication. (R3) the decentralization is achieved via the Sovrin network. (R4) Identity correlation can be prevented since the credentials, and the identifiers are independent of each other. Thus, an identity owner can use the same credential with different identifiers. (R5, R6) ZK capable credentials provided functionality for minimum data disclosure. Further, (R7) the SDK contains library wrappers for Java and iOS, and therefore, the wallet and functionalities are embeddable into a mobile app. (R8) The DIF identity hub could realize credential availability. The Indy community has proposed to integrate the hub into its sys-

No.	Requirement	Description
R1	Multiple identifiers	An entity should be able to use different identifiers depending on the use case.
R2	Identifier authentication	Two parties should be able to authenticate the counterpart's identifier
R3	Decentralized identities	Identities should be managed in a decentralized manner. No central authority, which has the power to create, edit, and delete the identities of the ecosystem.
R4	Non-correlating identities	It should not be feasible to correlate identities via public information or multiple interactions with the entity.
R5	Verifiable credentials/certificates	A digital credential, which is issued by an issuer and which can be verified from a third party. The credentials have to belong to a certain identity
R6	Minimal disclosure of personal information	Only necessary information of the entity should be disclosed.
R7	User own identifiers and credentials	The user should be in the centre of the IdM system to increase trust in Cardossier ecosystem.
R8	Credential availability	It should be possible that a third party can make use of a verifiable credential of an identity also when the entity (user) is not connected with the network.
R9	Revocable credentials	An issuer should have the power to revoke credentials.
R10	Issue credentials as group member	It should be possible to keep the entity of the issuer anonymous expect the information that he is a member of a certain group should be revealed.
R11	Key rotation	It should be possible to rotate keypairs at any time
R12	Data backup and recovery mechanism	Identities should be back up and recoverable such that the risk of data loss can be minimized.

Table 3.1: IdM system requirements

No.	Requirement	Description
R13	User consent for transactions	An user should be able to give consent to specific Corda transactions.
R14	Privacy preserving declaration of car ownership	The information of car ownership and the number of cars an identity owns should be kept private.

Table 3.2: Corda specific requirements

tem; however, the library is currently in development. (R9) There is a revocation registry implementation under zero-knowledge - meaning the identity owner makes a non-revocation proof. (R11) Key rotation is supported, and the process of updating the DID document in the ledger are implemented. Finally, (R12), an encrypted back up is supported and as a recovery mechanism, was a social recovery proposed.

3.3.2 uPort

We refer to Table 3.3, which gives an overview of the current uPort functionalities. uPort does not support our requirements (R4, R8, R9 and R10). However, uPort has an application in production, which supports several features. (R1) The concept supports it and can be realized by the provided libraries. (R2) A JWT based protocol is implemented for authentication purpose. Next, (R3, R4) verifiable credentials are realized on a JWT extended data format which allows to selectively disclose claims when for each claims a separate credential was issued beforehand. In case claims are bundled on one credential, the identity owner cannot perform selective disclosure. (R7) Identity ownership is enabled via a mobile app. Finally, (R11) key rotation is supported by rotating the public key in the DID document on the Ethereum network and (R12) is realized by seed recovery and a smart contract implementation of a social recovery mechanism.

uPort has an Ethereum based architecture. Thus, each mechanism, which has a smart contract involved requires an Ethereum transaction such as key rotation, social recovery and publishing as well as updating a DID document on the ledger. In order that an identity owner does not require some Ether, uPort has an implementation that allows a third party to pay the transaction fees.

3.3.3 BlockCerts

By referring to the overview in Table 3.3, the BlockCert application is evaluated. BlockCert is the less featured project. It does not fulfill the requirements (R2, R4, R6, R8, R10 and R11). For completeness, we briefly mention the key elements. BlockCert currently does not support DID's, but instead, the (R1) identifiers are the (R3) Bitcoin or Ethereum addresses. Further, (R5) it supports certificates; however, they are no selective disclosure of claims. For storing

and managing the purpose of the certificates (R7), a mobile app is provided. Finally, (R9) is enabled via a revocation registry and (12) is achieved by back up to external storage such as Google Drive or Dropbox as well as seed recovery for private keys.

3.3.4 Conclusion of Evaluation

The evaluation has shown that the toolkit of the Hyperledger Indy project enables us to fulfill the most requirements. Especially, the independence of identifier and credentials is an essential property to protect the identity owners privacy and to prevent from identifier correlation. In consequence, the Indy should be used to realize self-sovereign identities. There are three options providing SSI through the Indy-SDK. One option is to participate in the Sovrin network and use it as decentralization layer for the decentralized public key infrastructure (DPKI). An option in the mid-term when the potential of SSI can be exploit in Cardossier and the decision is reasonable from a business point of view. A second option is to set up a Indy-based network next to the Corda network. However, it cause a large overhead to operate and maintain two networks. A third and last option is to integrate a DPKI which is compatible to the Indy-SDK. The third option should be realized since the DPKI can be implemented on Corda within reasonable effort.

No.	Hyperledger Indy / Sovrin	uPort	BlockCerts
R1	DID's by wallet [indy-sdk, indy-plenum]	DID's by [ethr-did]	Bitcoin/Ethereum address
R2	Authentication via challenge [indy-sdk]	JWT based authentication. [uport-credentials, uport-transports]	Not supported
R3	via Sovrin network	via Ethereum	via Bitcoin or Ethereum
R4	Identifiers and credentials are independant cause of ZK verifable credentials [indy-sdk]	Not supported	Not supported
R5	Zero-knowledge capable verifiable credentials. [indy-sdk]	JWT credenitals [uport-credenitals]	Yes, JSON-LD credentials. [cert-issuer, cert-verifier]
R6	It allows to hide claims of a credential and to create minimum/maximum proofs [indy-sdk]	Allows to selectively disclose claims [uport-credentials]	No, always the full credential is presented [cert-verifier]
R7	Java (Android) and iOS sdk wrapper. [indy-sdk]	mobile app [uport-mobile]	mobile app [wallet-android, wallet-iOS]
R8	DIF identity hub [agent proposal]	Not supported	Not supported
R9	non-revocation proof [indy-sdk, indy-plenum]	Not supported	Revocation registry [cert-verifier]
R10	Not supported	Not supported	Not supported
R11	Wallet master key and DID key pair rotation. [indy-sdk, indy-plenum]	Supported [ethr-did-registry]	Not supported, no DID document
R12	Encryped back up & social recovery [agent proposal]	Seed recovery [uport-mobile] or scoial recovery [uport-identity] for identifier	external back up and seed recovery for identifiers [wallet-android, wallet-iOS]

Table 3.3: SSI projects evaluation on basis of our requirements
Square brackets: software library

CHAPTER 4

Concept Proposal

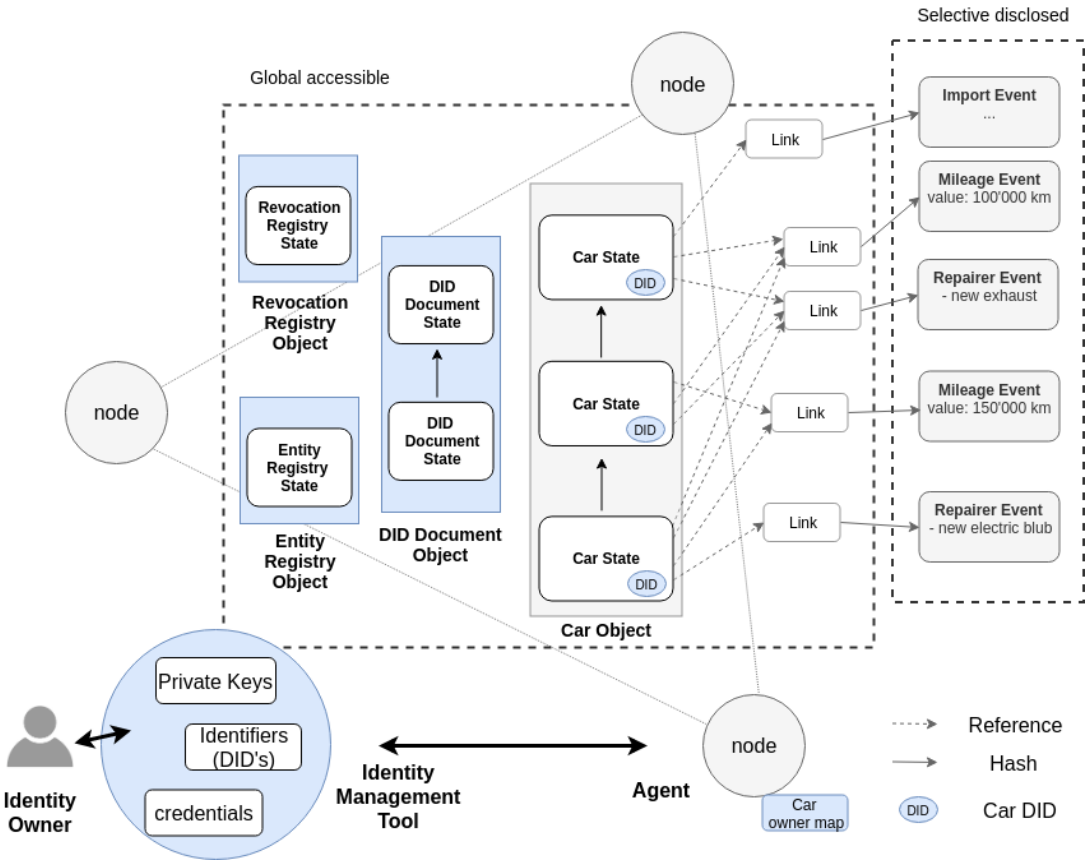

Figure 4.1: An abstracted model of Cardossier. It shows all existing components in grey while the extended one's are marked as blue.

The proposed concept and considerations are discussed in the first section of this Chapter. In the second section, we present a system architecture to implement the proposal.

4.1 Data Model

This section refers to the model illustrated in Figure 4.1 by discussing the individual elements of the model.
The essence of the model depends on Corda properties, which differ from traditional blockchains. However, we will not go into greater detail of Corda unless it has a direct impact on the design. As a general introduction to Corda see Section 1.3.

4.1.1 Car Data Model - Existing Model

In Cardossier, all data provider place car-related data on the network. Since several providers will input data for the same car, a shared car identity is required. In Cardossier it is done by allocating a Corda state object 1.3.1, called car object, for each car, see Figure 4.1. The car object is shared with the notary and all involved parties. Since it does not contain any car data nor personal information, only a car ID and link object references, it is considered as non-critical data. A link object is an intermediate between car states and an event to hide the events from the public and allow selective disclosure. For each specific purpose, there is a distinct event type; however, in general, they contain car related information. All these events are stored at the agent which the car owner favours most.
This design has several advantages: (1) it is privacy-aware. The car-related data are kept locally under the control of the agent, which can selectively share events with other participants via Corda Flows. (2) Events are globally verifiable. The link object contains the hash of the event and the Car State a reference to the link object. Thus, any third party can verify that an event belongs to the Car State. (3) Data completeness is verifiable. When he becomes the new agent, all events have to move to him. He can be verified that the number of link states receives all events. (4) It is scalable. The complete data of the ecosystem is segmented in car objects and events. A car object never depends on other car objects, and thus, only a single car object is required for data verification. Additionally, a car object is only shared between involved parties and events only between the event issuer and the agent. Network participants only have to manage a fraction of the network data.

4.1.2 Identity Data Model

In this subsection, the extended components to enable SSI in Cardossier are presented. The component descriptions below relate to Figure 4.1

Private Keys

A decentralized public key infrastructure is established. The identity owner controls several private keys to create signatures and zero-knowledge proofs.

Identifier (DID)

Each identifier is a DID, and thus, an identifier belongs to a public/private keypair. It allows to prove ownership, and it enables to authenticate against any service or entity.

Credentials

A credential is a collection of claims which belong to the identity owner. Credentials are not bounded to a DID but instead to a secret (private key) of the identity owner, what allows to prove claims under zero-knowledge. A verifier can verify each claim without contacting the credential issuer.

Identity Management Tool (IMT)

The identity owner has to be able to store his private key and manage his identifier on a private wallet. For this purpose, the use of a mobile app is proposed. It should allow the identity owner to control its identity and to interact with agents, credential issuers and verifiers.

DID Document Object

The identifiers corresponding public key is stored in an DID document object[1], see Figure 4.1. The object is considered non-critical since it does not contain any personal nor personal related data. It provides a map from the identifier to the public key. Since the public key must be globally accessible, the object is shared between the agent and the notary. It allows us to resolve every DID via the notary to get the DID document state. The control of the DID document object is given to the identity owner who has to sign each update with its private key. Every node of the network should be an agent, and therefore, the identity owner is independent of the agent. As long as the notary is a sufficient decentralized network, it can be trusted, and thus, the identifier belongs to the identity owner.

(Credential) Revocation Registry Object

A global registry, which defines if a verifiable credential is still valid or not. It also shows when a credential was issued. The issuer can revoke the credential and reverse it.

[1] A Corda state object 1.3.1

Entity Registry Object

The entity registry is a shared object between the notary and the Cardossier association. Each organization participating in the network with a Cardossier node is registered at the association, which provides a map between organization information and organization DID in the form of the entity registry. Only the association should be able to create, edit or delete an entry in the registry. It is achieved by giving only to the association node permission for these purposes. Thus, the correctness of the registry is validated by the association node and the notary. However, the registry can be resolved by all nodes of the network.

Car DID

A car DID declares car ownership. The identity owner, which owns the car, has the private key of the DID in his identity management tool. It allows proving ownership to any node or other users via a challenge-response scheme.

Car Owner Map

A registry at the agent containing a map between card ID an a DID of the actual owner. This DID is never shared within the network. Its only purpose is to provide a fast query mechanism in case of large car fleets. Without it a car owner have to prove ownership for each car separately.

4.2 Features

This section describes the functionalities of the proposed concept.

4.2.1 DID Authentication

The authentication takes place via challenge-response. After a request from the identity owner, the verifier creates a challenge containing a random nonce. The identity owner adds a DID to the challenge and signs it with its private key. The verifier verifies the response via the public key on the DID Document.

- **DID document object**: It contains the public key
- **Identity management tool**: It contains the private key.

4.2.2 Multiple identifiers

Every identity owner can create as many DID as desired via its identity management tool. Creation of a new DID also enforce the creation of a keypair whereby the public key is transmitted to the agent and globally published as part of the DID document.
It is realized by:

- **DID document object**: A new document object is create for each DID
- **Identity management tool**: Allows to create new keypairs and DID's.

4.2.3 User owns Identifiers and Credentials

It also implies decentralized identities. Each user owns his identities in the sense that he and only he knows a secret. The knowledge of this secret is equivalent to the ownership of the identifiers and credentials. It requires that the identity owners have their private keys on their devices.
It is provided by:

- **Identity management tool**: A user device which stores private keys and credentials.

4.2.4 Verifiable Credentials

The credentials are zero-knowledge capable verifiable credentials, and the identity owner owns them. Each credential contains a master-secret-binding to the owners private secret. This secret allows an identity owner to create four kinds of proofs: (1) he is the owner of the credential without revealing its identifier, (2) he has a credential attribute without revealing it, (3) he can selectively disclose credential attributes and (4) he can prove that an attribute is smaller or larger than a number. Furthermore, he can self-assert claims to the proof.
Any third party can verify each credential without interaction with the issuer. For the verification process, the verifier requires the public key of the issuer to check the signature. Each credential includes the issuer's DID which is resolved to get the public key from the DID document state. Furthermore, the identity of the issuer is requested from the Entity Registry Object. It allows us to decide the trustworthiness of the issuer. Additionally, there is a level of assurance provided, which indicates the quality of the credential.

Level of Assurance:

- State 1: There is minimal confidence. The holder has only be authenticated, and the data might be correct.
- State 2: There is some confidence. The holder identity has been verified form a State 3 or higher and the data might be correct.

- State 3: There is high confidence. The holder identity has been verified from a State 4 credential, and the data is correct.
- State 4: There is very high confidence. The holder entity has been verified, and the data is correct.

With this two information, the issuer's identity and the level of assurance, the verifier can decide if he accepts the credential or not.
It is enabled by:

- **Anonymous Credentials**: verifiable credentials are specified by W3C and an zero-knowledge capable concept was presented by Hyperledger Indy.

4.2.5 Minimal Disclosure of Personal Information

Identity owners privacy should be preserved. For instance, only the actual requested data should be presented to a verifier. These are achieved by ZK capable verifiable credentials, which enable to hide claims in a verifiable presentation, or only to prove the existence of a claim. Additionally, it allows proving that a claim has a value large, smaller or equal a certain number. Besides, the user should be able to store his credential on a local device. Otherwise, he has to trust on a third party for storing his personal information (credentials).
It is achieved by:

- **ZK capable verifiable credentials**: It allows to hide not required claims of a credential, to prove only the existence of a certain claim and to compare a claim against a number without revealing its actual value.
- **Identity management tool**: A user device which stores private keys and credentials.

4.2.6 Credential Revocation

Three approaches can enable the possibility of the issuer to revoke issued credential. (1) time-revocation: credentials contain an expiration time, (2) revocation list: credentials are linked to an index of a revocation registry, which can only be updated by the credential issuer, or (3) a proof of non-revocation: credentials include the zero-knowledge capability to prove that the credential has not been revoked. This proof does not reveal any credential identifier as it is the case with a revocation list, and thus, a verifier does not be aware when different claims from the same credential have been presented.
All three options should be considered depending on the use case. A time-revocation to indicate a expire data such that the identity owner and verifier know the validity time range, e.g. for a road permission. A revocation list is required to revoke embedded presentation, see Section 4.2.9. It can be realized via a revocation registry object in Cardossier. Only the owner of a registry will be able to modify it - in this case, only the issuer of a credential. Ownership

is ensured through a smart contract and the notary, similar to the DID document object. Finally, we propose to use proof of non-revocation of general use cases of verifiable credential since it enables great privacy by preventing credential-correlation.

- **Time-revocation**: An expired data part of the credential
- **Revocation list**: A map between credential ID and revocation state
- **Proof of non-revocation**: Zero-knowledge proof that the credential has not been revoked - a concept of Hyperledger Indy, see Section 2.2.1

4.2.7 Anti-Correlation

It is desired that privacy also remains after many interactions and information cannot be gained through clustering or correlation. There are two elements which should remain private: (1) which cars and the total number of cars an identity owner owns and (2) the activities of the identity owner on the network. In essence, both can be achieved when supporting different identities for each activity and car ownership. Therefore, it should be supported that an identity owner can have multiple identifiers and create new ones whenever desired and that claims do not rely on an identifier. The credentials and identifiers remain independent. It has the effect that credentials can be used in combination with any identifier. Additionally, unique credential identifiers (DID) should not be revealed to the verifier unless necessary.
Two features are supported to prevent form correlation:

- **Multiple identifiers**: A user can have several identifiers (DID's) and create new ones whenever necessary.
- **Anonymous credentials**: An user providing a credential to a verifier does not have to reveal its identifier. Instead, he can prove credential ownership in a zero-knowledge proof.

4.2.8 Credential Availability & Data Sharing

It is necessary for Cardossier that credentials can also be issued when the identity owner went offline. For instance, after the identity owner and the issuer have interacted, credential issuance may have to be triggered manually since some offline work must be done before. Therefore, the functionality of a credential "inbox" should be provided, similar to a mailbox. The identity owner can select an inbox service provider. The address of the inbox is then written on the DID Document by the IMT. However, it is important that the inbox address is different for each DID, otherwise, DID correlation is feasible. Furthermore, the issued credential should be encrypted by the issuer with an identity owner public key of the DID Document.
In the same way, data sharing can be achieved. With the difference of supporting more functionalities such as giving data access to other parties selectively with reading and/or write

access.
Availability and data sharing is achieved by a credential inbox and a hub service:

- **Credential "inbox"**: A service provided by an agent similar to a mailbox, but for encrypted credentials.
- **Hub**: It allows to store and shared signed data on a third party data storage (cloud) while using DID authentication and authorization.

4.2.9 Issue Credentials as Group member

An issuer should stay anonymous except for knowing that he is a part of a particular group. E.g. an identity owner want to prove that a certificate of insurance has been issued from an accredited insurance company without stating the company. It is achieved by embedding a verifiable presentation, which demonstrates the group membership, to the issued credential. Beforehand, the issuer must have received the membership credential from an authority. It could be the Cardossier association acting as a trust anchor. The membership credential is a regular credential containing the association's DID and a claim "insurance company". First, the issuer creates a new credential definition (ZK public key), which he self-asserts to a verifiable presentation proving the membership. This presentation is added as a claim to the new credential. There is no adaption for the identity owner except that he should verify the membership proof. Next, the identity owner creates a regular, verifiable presentation and presents it to a verifier, who finally verifies both presentations, whereas only the embedded presentation contains a DID (association DID). The self-asserted credential definition establishes the link between the two proofs. Since only the owner of the master-secret of the embedded credential can create a verifiable proof and only he knows the private key of the Camenisch-Lysyanskaya signature of the issued credential.
A verifiable credential with membership proof will have the following structure:

```
{
    // pseudo code
    ...
    "claims": {
        "embedded presentation":{
            "DID": "did:cardossier:associationNode",
            "license": "car insurer",
            "credential definition": {...},
            "proof":{...}
        },
        claim1,
        claim2,
    }
    "signature": {...}
}
```

However, this approach comes with a limitation in credential revocation of the embedded credential. Proof of non-revocation cannot be used for embedded credentials since the accumulator value might have changed when a presentation of the outer credential is verified. Therefore, a revocation list must be used for embedded credentials. It can is enabled by:

- **Embedded verifiable presentation**: A verifiable credentials which contain a proof from the issuer. It allows us to prove issuer properties to the verifier via the identity owner.

4.2.10 Keys & Data Recovery

We present a backup and recovery mechanism which rely on Shamir Secret Sharing. Furthermore, we discuss how to recover from key compromise and security risk including its mitigation.

Backup Mechanism

Still, an open challenge in the field is to provide a proper recovery mechanism for identity owner's secrets. The private keys, as well as all credentials and identifiers, should be backed up. One way is to store an encrypted backup on an external system selected by the identity owner. It has low usability since the responsibility is left over to the identity owner. It cannot be ensured that he manages the backup properly. Another way is a social recovery mechanism. The wallet is secret-shared with trusted friends and family. However, it presupposes that they all have an application on their phones and that there are no strong social conflicts. We present another approach, also based on secret sharing, but the share is distributed along with some nodes. A network map provides a list of the IP addresses of the nodes. It allows selecting nodes (backup nodes) based on a deterministic random function with the identity owner's email address and an arbitrary number (R) as input. The number is sent to the email address as a backup. Authentication of the identity owner is achieved by email and password or by FIDO authenticators. In this setup, the identity owner can recreate the ID's of the backup nodes with R and the mail address. Additionally, the password can be reset by the nodes via an email. The risk of being compromised via email compromise could be reduced by enforcing a second authentication factor for password changes.
The presented backup mechanism can provide additional added value. The wallet can be reconstructed from anywhere and on any device only with knowing a secret. It allows us to run an implementation of the IMT in the web browser without remote hosting of the wallet. The in-browser IMT collects the shares from the nodes and reconstructs the wallet locally.

- **Shamir Secret Sharing**: Sharing the wallet along multiple nodes based on user secret.

Key Compromise

There are three kinds of key compromise. In case of a wallet compromise, all keys should be rotated.

1. A DID can be compromised, meaning its private key. The key can be rotated and the DID document object being updated, though not when already done by the attacker. Therefore, the backup nodes should have the power to rotate the keys via an multi-signature transaction. Only when a certain threshold of backup nodes sign the transaction the DID document get updated. To do so, the DID document must contain the public keys of the backup nodes. Preventing that the attacker can change the listed backup nodes on the DID document, updates should only get valid after a particular time frame, and the identity owner should be informed about the change via an email from the backup nodes. It allows the identity owner to react on not consented changes.

2. After a master-secret compromise all credentials belonging to the secret should be revoked, and a new secret be created. Otherwise, the attacker can make use of the credentials and impersonate the identity owner.

3. In case the password for authentication against the backup nodes are compromised, the identity owner can rotate the password with his email address as described in the Paragraph Backup Mechanism.

Security Consideration

There are two types of attacks - active and passive attacks.

Active attack - the aim of the attack is to take over identifiers and/or credentials such that the identity owner can not recover from:

- DID compromise. The attacker rotated the keys as well as altered the authorized backup nodes. The new backup nodes became valid after the time frame expiration since the true identity owner did not react.

  ⇒ The time frame should be of, and it must be guaranteed that the notification reaches the identity owner to minimize the risk.

- master-secret compromise. The attacker can not rotate it, and thus has no risk for an active attack.

- wallet compromise. A threshold number of backup nodes become compromised or start to act maliciously. They can take over all identities by reconstructing the wallet and rotate all DID keys. In this case, an identity owner will not be able to recover since the backup nodes do not act in his intend.

  ⇒ A high threshold can minimize the risk. However, it has to be carefully selected since it has the consequence that the data storage rises. Nevertheless, the threshold may be set to a rather small value since all nodes are well-known and registered. The node operators can be made accountable.

Passive attack - the attacker stays passive in the sense that he does not rotate keys:

- DID compromise. It would allow the attacker to authenticate against service providers and potentially to access personal information.

  ⇒ Providers should trace the user access and inform the identity owner about unusual activities via a second channel, e.g. messaging service on the DID document.

- master-secret and credential get compromised. Besides the attacker got personal information, he can make use of the credential to create presentations. The attack is hardly detectable as long as no damage for the identity owner becomes visible.

  ⇒ Minimizing the risk by wallet encryption with state of the art cryptographic schemes.

4.3 SSI Utilization in Cardossier

We demonstrate three cases to utilize self-sovereign identities.

4.3.1 General Utilization

The verifiable credentials and DID's of the SSI ecosystem are used to manage personal data in Cardossier. See Section 3.1 for more information about the Cardossier data types.

4.3.2 User Consent for Transactions

In some case, it is desired that an identity owner can sign a transaction, or in other words, give consent for a transaction. It can be realized by a signature on the transaction input values. The user/identity owner has a private key in his wallet, whereas the public key is part of the state object in the network. When it comes to a transaction, the notary of the shared object has to verify the signature. The verification is defined in the smart contract. Since the notary is a decentralized entity, it can be assumed that the notary behaves correctly. Thus, only transactions with a valid input signature of the identity owner are approved and executed.

4.3.3 Car Ownership and Transfer of Ownership

How can car ownership be claimed in a privacy-aware and scalable manner? On the one hand, the owner should be able to prove ownership to any third-party in a way that does not allow to correlate several cars to one identity (assuming owning several cars). Moreover, on the other hand, the agent should be able to query all cars of the identity owner in a fast way. It is of importance when it comes to fleet management. Both should be supported, and therefore, two ways of proving ownership should be provided. The owner DID should be

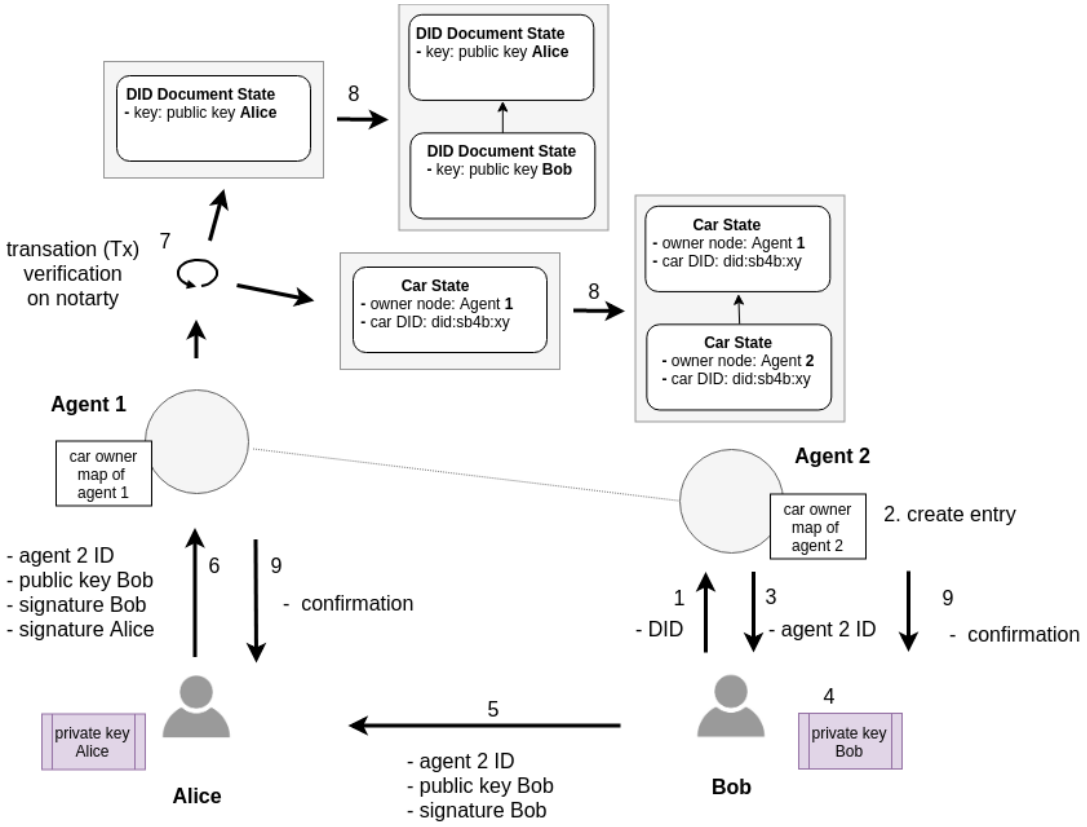

Figure 4.2: Transfer of ownership

stored in an owner map object on the agent and never be published to another party. It is an internal mapping between car object and DID. Proof of ownership to third-parties and can be made via an car DID. The car owner is in control of the DID - he owns the DID private key. Besides, the key is used to sign the change of ownership transaction.

- **Car owner map**: When it comes to large car fleets, all car states can be queried in a fast way with one single DID.
- **Car DID**: A DID which is controlled by the car owner. The owner can prove ownership to a requester by responding on a challenge.

Transfer of Ownership: Figure 4.2 illustrates the process. Initially, the car data is managed by agent 1 and Alice is the current car owner since she controls the keypair A. Bob should become the new owner. Agent 1 and agent 2 have already exchanged the car state and events such that Bob could watch the car data. The transfer is realized in eleven steps:

1. Bob demonstrates DID for car ownership to agent 2
2. Agent 2 creates entry in car owner map (DID, Car DID).
3. Agent 2 returns its ID; used by agent 1 to identify the receiving node.
4. Bob creates a new key pair (public/private key) and stores it in the wallet.
5. Bob signs process ID and public key with his private key, and he transmits the data to Alice
6. Alice verifies correctness and signs the data. Alice triggers a Corda transfer of ownership transaction via the agent 1
7. Agent 1 performs transaction flow. Notary verifies the signatures and data correctness
8. A DID key rotation is performed to give control of the car DID to the new owner Bob. Moreover, a new car state is created and the car object updated with a new owner node.
9. Agents confirm transaction completeness to Alice and Bob

4.4 Credential Ecosystem

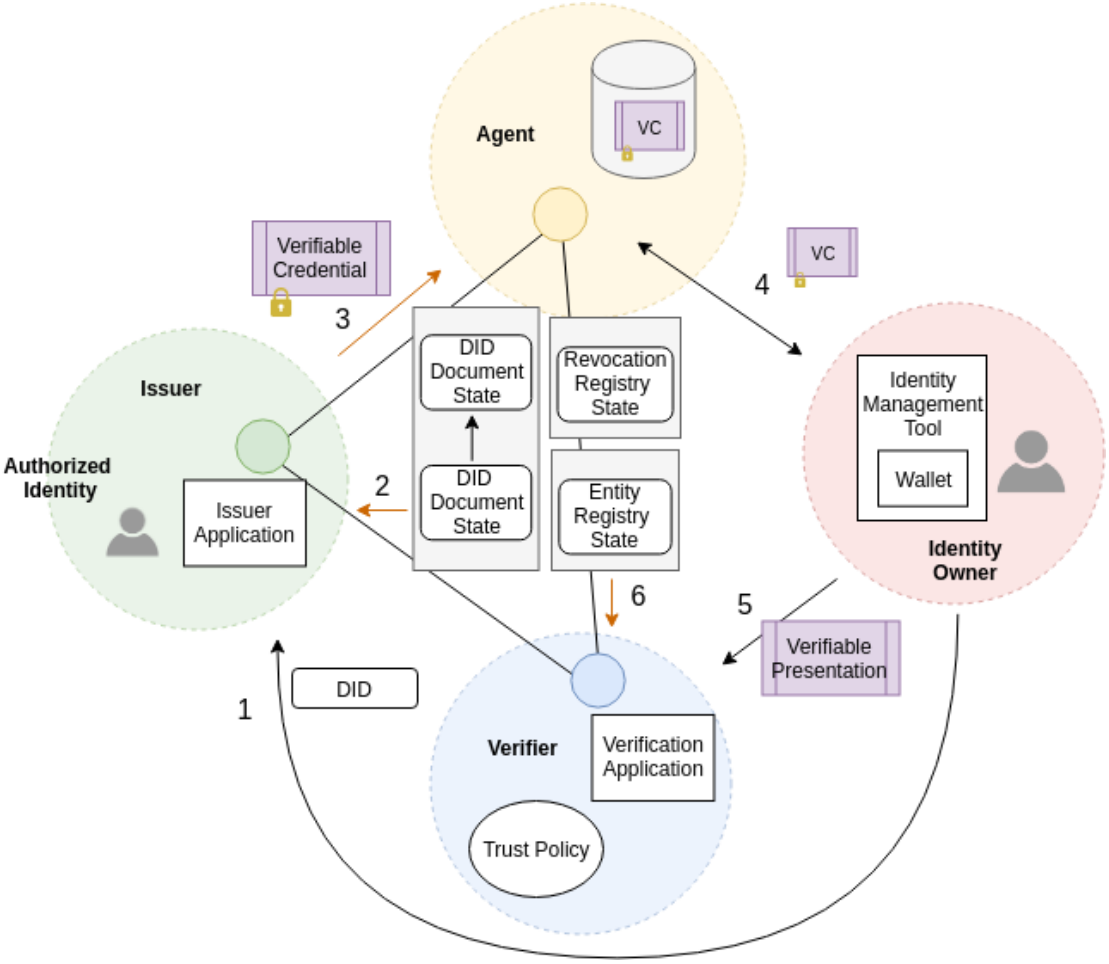

Figure 4.3: Credential ecosystem: from the creation to the utilization of a verifiable credential

Figure 4.3 is on the center of this Section. It illustrates the proposal of a credential ecosystem in Cardossier. In the first part, we look at it from a conceptual perspective, whereas the software architecture of each of the four roles is described in Subsection 4.4.1. Also, the interactions between roles as well as between architecture components, are stated out in the Subsection 4.4.2.

The credential ecosystem consists of four roles, the credential issuer, agent, identity owner and verifier, whereby the identity owner is the central player and at the start of most interactions. By discussing Figure 4.3, (1) the identity owner initialize the interaction with the issuer, exchange credential request data and provides a DID via DID authentication, (2) what requires to resolve the DID for the corresponding DID Document. Also, the issuer may request a secondary authentication to confirm an existing identity or request for identification via an

official identity document. After it, the identity owner may go offline, since the claims, which is going to be issued in the form of a verifiable credential, has to be approved manually. In the next step, (3) the issuer creates a credential, encrypts the credential with a DID's public key and transmits it to the agent. At this point, it has to be mention that the credential inbox address should be part of the DID Document. Thus, the issuer knows the address after resolving the DID. Further, (4) the identity owner goes only, requests the credential from his inbox and stores the credential in the identity management tool (IMT). Now, (5) he can make use of the claims in a verifiable presentation, which is going to be shown to the verifier. (6) The verifier resolves the issuers DID from the entity registry to get the identity of the issuer, and he requests the credential revocation state from the revocation registry. Moreover, he verifies the signature of the presentation and applies the trust policy, which defines which issuer he trusts and which level of assurance of the claims is required to accept the presentation.

4.4.1 System Architecture: Roles

The software architecture of each role of the credential ecosystem, Figure 4.3, are provided in this Subsection:

Issuer

An issuer can be any organization running a Cardossier node. Each issuer is well-know and registered at the Entity Registry. In Figure 4.4, the building blocks of the issuer application is shown. It consists of access management at the top to control the access. The next layer is the management board, which is a user interface with the key functionalities of triggering credential issuance, revoking credentials and viewing issued credentials. One layer below there is the backbone of the application. The logic layer handles the user actions and coordinates between the modules: (1) a credential store is a database to store all issued credentials, (2) the wallet holds the issuers private keys and creates the verifiable credentials, (3) the service API is the interface to communicate with the identity owner and (4) the network adapter provides the connection to the Cardossier network and components, see Section 4.4.1.

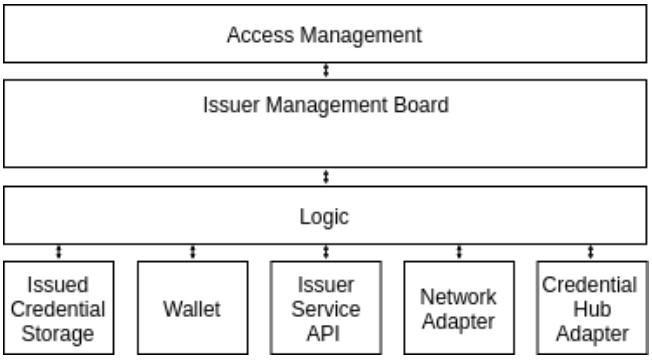

Figure 4.4: Issuer architecture

Agent

The agent offers several services to the identity owner application. The hub service (left) allows the identity owner to store its credentials in the cloud as well as to share and receive credentials when offline. Its building blocks are a database to store the credential (credential storage), an access management module and the hub API, see Figure 4.5. A second service offered by the agent is the network service (centre) to provide access to Cardossier network components such as DID Resolver or Entity Registry. Additionally, the agent provides a backup service via secret sharing to the identity owners. It consists of a database to store the shares, an authenticator and the API. Its structure is similar to the hub expect that the authenticator module is more restrictive than the access management module. It does not allow to share data nor to authorize access.

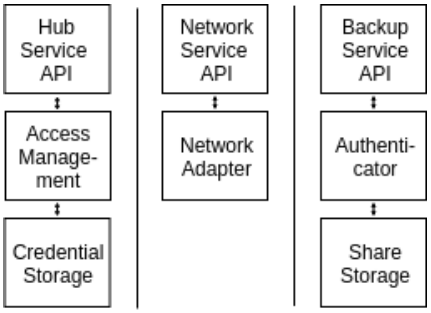

Figure 4.5: Agent architecture

Identity Owner

Each identity owner has an Identity Management Tool (IMT) to control his identities. The top layer of an IMT is the access management followed by the management board, which allows the identity owner some key functionalities: (1) to view and delete credentials, (2) to create, view and delete identifiers, (3) to rotate keys, (4) to backup and recover its wallet, (5) and to select claims and identifiers for proving its identity to a verifier, see Figure 4.6. Further, a logic layer handles the identity owner interactions and coordinates between three modules: (1) the wallet connector to communicate with verifiers and issuers, (2) the wallet to store identifiers and credentials as well as to create verifiable presentations (proofs) and (3) an agent adapter to connect with services provided by the agents.

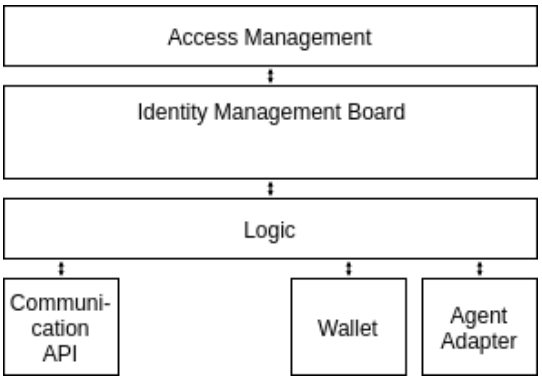

Figure 4.6: Identity Management Device

Verifier

The verifier checks the verifiable presentation provided by an identity owner and decides the trustworthiness of the data. Building blocks are a network adapter to access the DID Document and Entity Registry, tools for signature verification and a trust policy, which defines the trusted issuers and the required level of assurance for specific claims. On top of that, there is a logic layer and interface, see Figure 4.7.

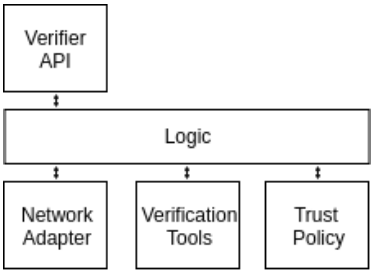

Figure 4.7: Verifier architecture

(Corda) Network Adapter

The network adapter provides access to distinct objects in the network, and thus, requires a Cardossier node. It allows to access the DID document object's and to update them when permitted, or also to create new ones. Further, information about the issuer can be retrieved from the Entity Registry. The Revocation Registry contains a reference of each credential and a mark when the issuer has revoked the credential. Each node can check the revocation state.

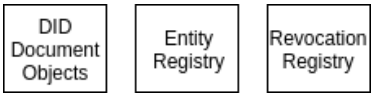

Figure 4.8: Network Adapter

4.4.2 System Architecture: Interactions

In this subsection the interactions between the architecture components of each individual role as well as between the roles are described in form of sequence diagrams:

- **Issuer**: interaction with the identity owner, see Figure 4.9
- **Agent**: interaction with the identity owner, see Figure 4.10
- **Identiy owner**: interactions with the issuer, agent and verifier, see Figure 4.11 and Figure 4.12
- **Verifier**: interactions with the identity owner, Figure 4.13
- **DID-Authentication**: challenge-response interaction, see Figure 4.14

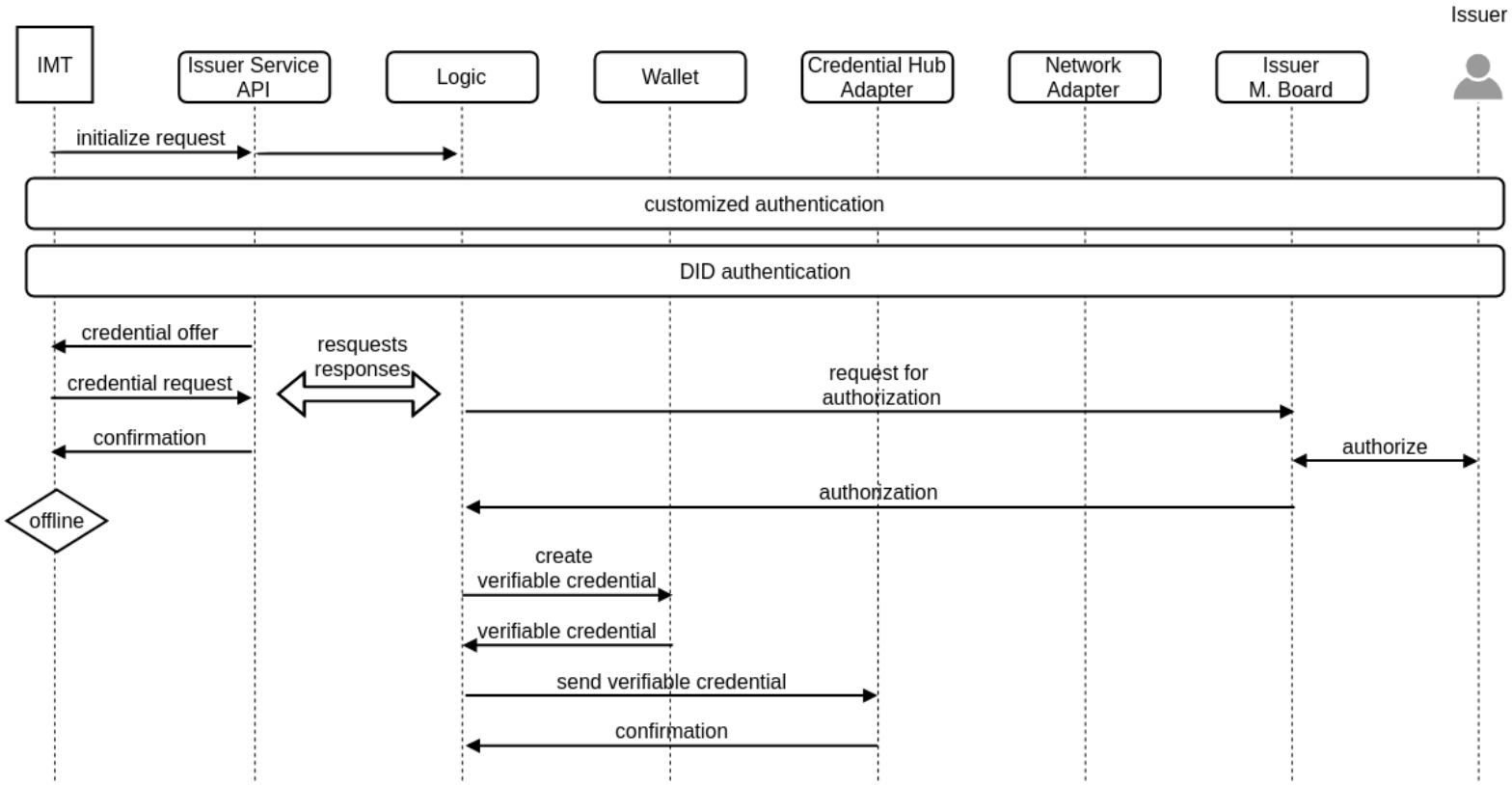

Figure 4.9: Sequence diagram: issuer

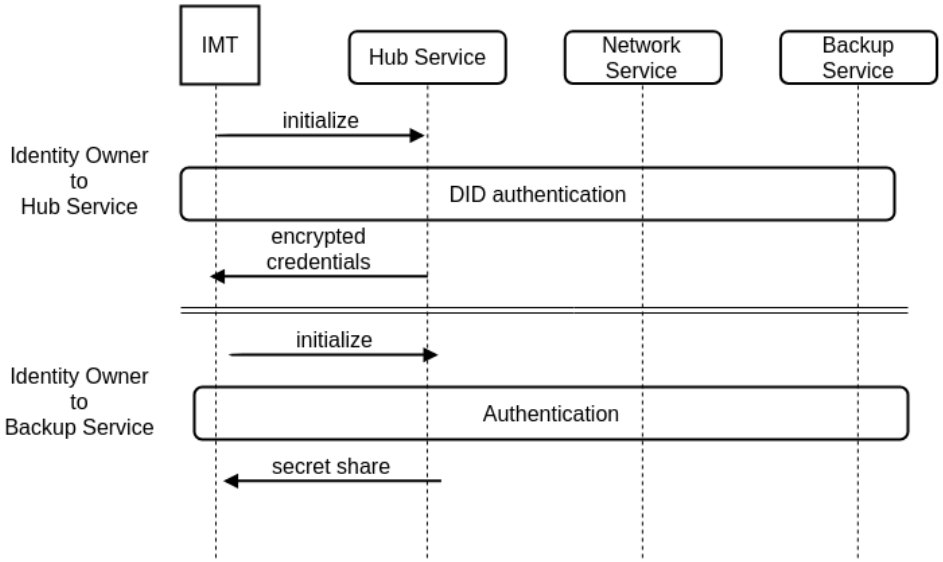

Figure 4.10: Sequence diagram: agent

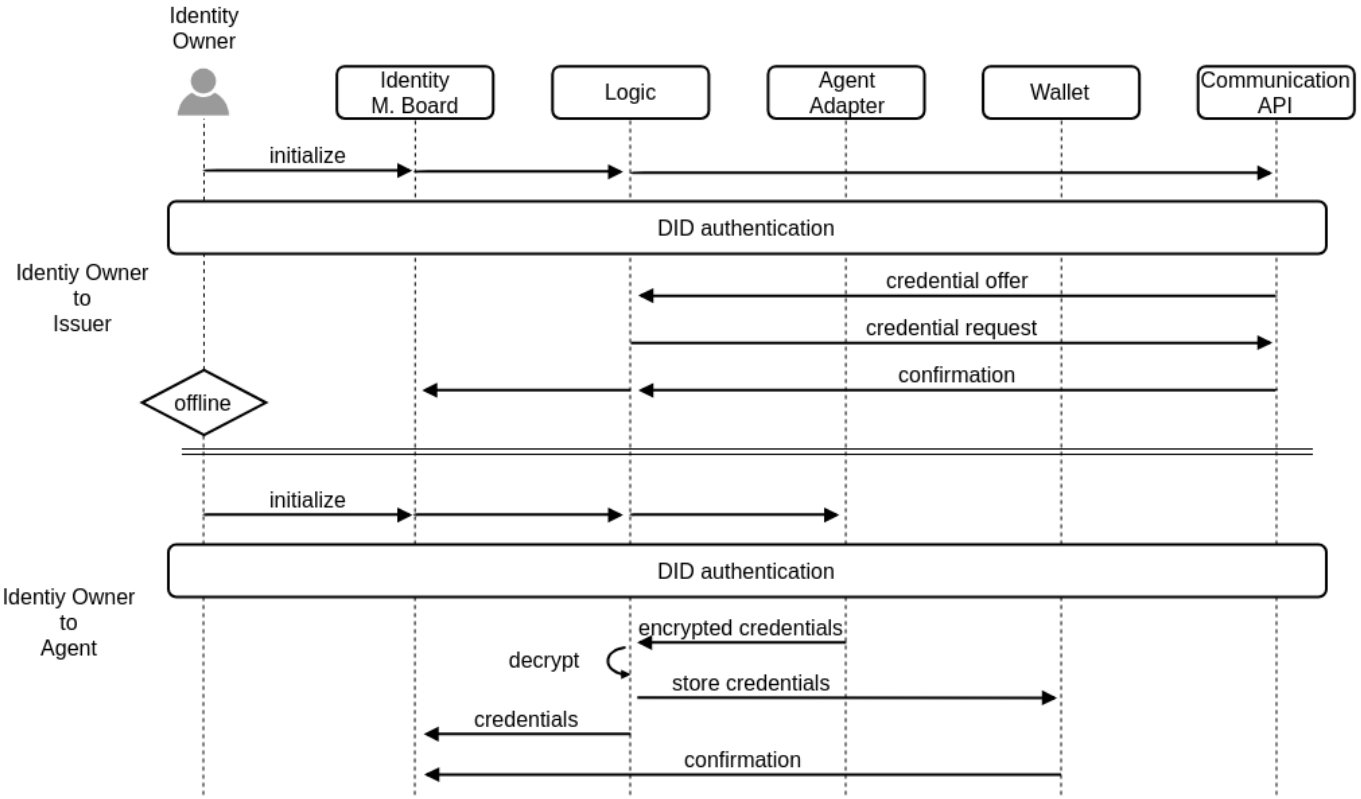

Figure 4.11: Sequence diagram: identity owner

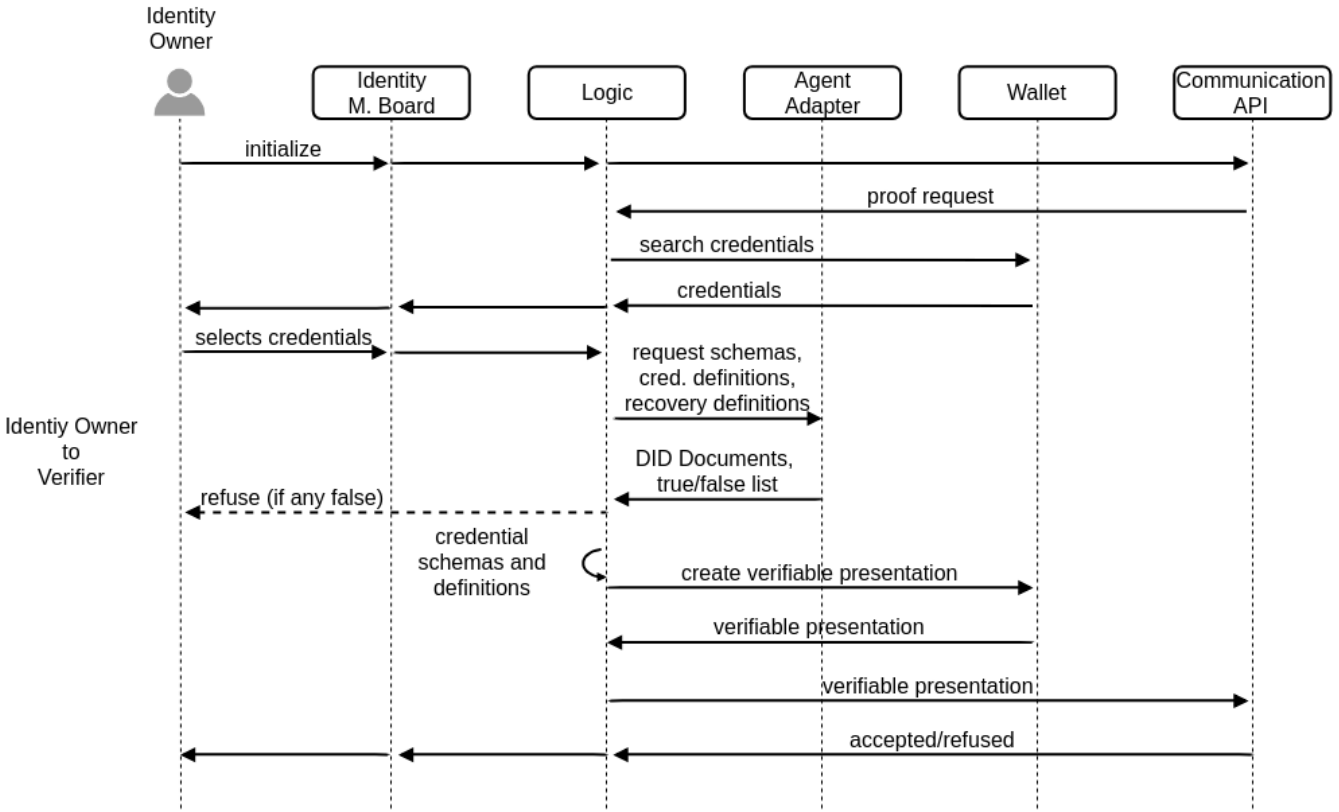

Figure 4.12: Sequence diagram: identity owner

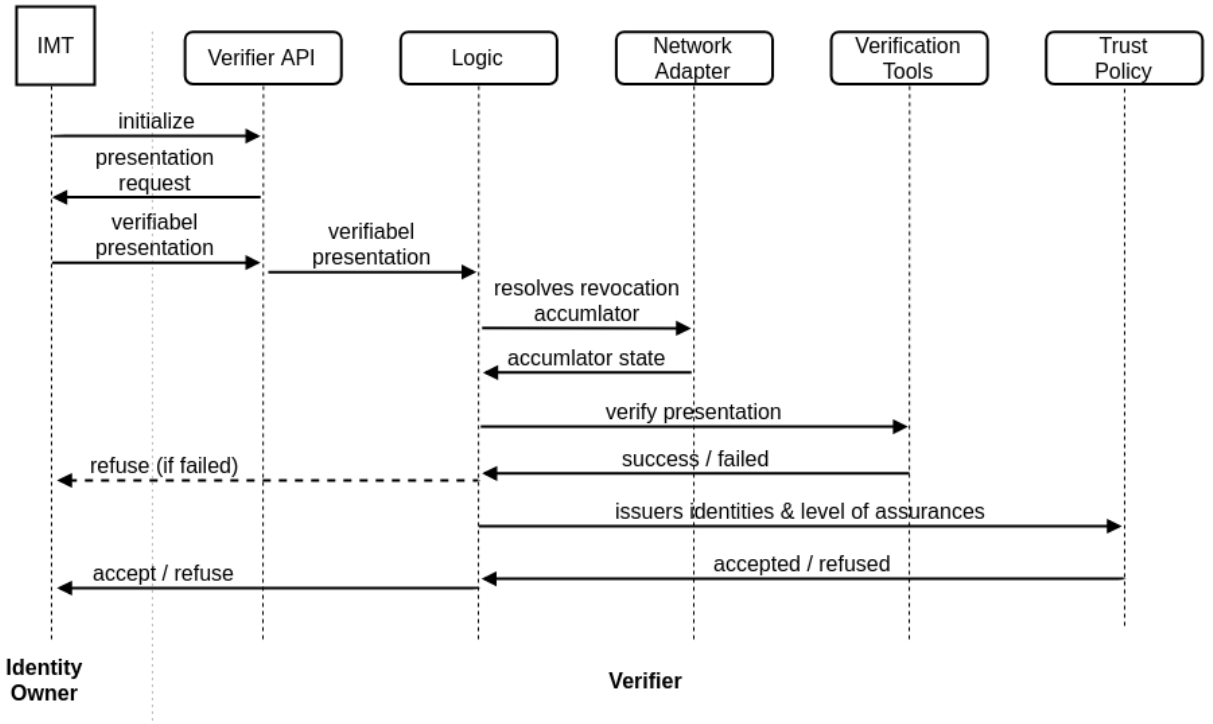

Figure 4.13: Sequence diagram: verifier

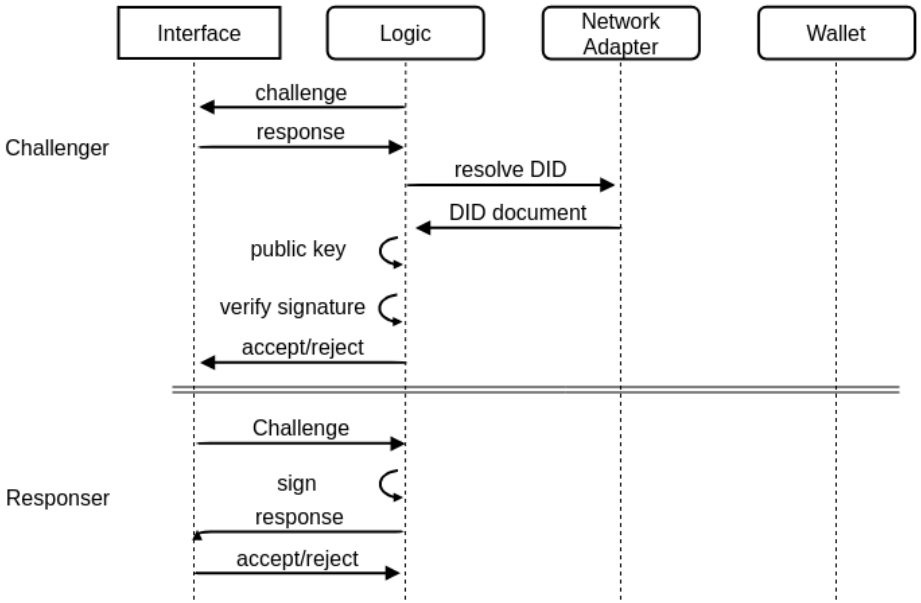

Figure 4.14: Sequence diagram: DID authentication

CHAPTER 5

Implementation

5.1 Tools and Libraries

A general view on the technology stack: R3 Corda is used as distributed ledger and mainly written in Java. Java is the main programming language in Cardossier combined with Lombok and Spring Boot. TypeScript and Angular are applied for web development. As build tool, Gradle is used; For deployment, Kubernetes.

The wallet implementation and tools from the SDK of the Hyperledger Indy project are used [30]. It provides the following functionalities:

1. **Manage private keys:** the wallet manages the DID private keys (signing keys) and a master-secret which is required to perform zero-knowledge proofs.
2. **Wallet encryption:** an user can provide a wallet key to encrypt the wallet. Additionally, the key is used in deriving DID keys.
3. **Create a DID:** the creation of multiple DID's is supported. The new DID including its private key is automatically registered at the wallet.
4. **Store DID with meta data:** additional to the DID meta information can be stored
5. **Create master-secret**: the secret is stored in the wallet and it is identified via a secret ID.
6. **Rotate the wallet key:** the user can change the encryption key of the wallet.
7. **Rotate DID keys:** a new keypair can be created and directly registered at the wallet.
8. **Store verifiable credentials:** all credentials are stored in the wallet.
9. **Query verifiable credentials:** it allows to search for credentials with specific attributes.
10. **Create ZK verifiable presentations:** the master-secret is required to create ZK-proofs.
11. **Store additional data:** can be stored via tags.

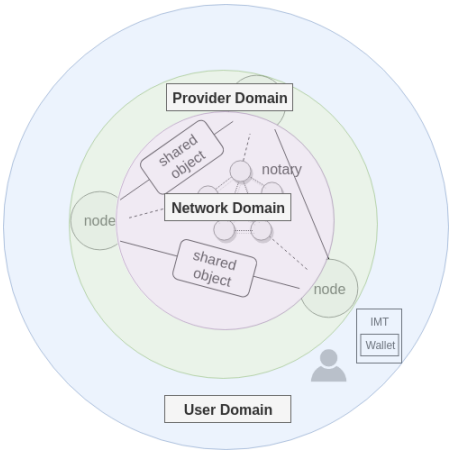

Figure 5.1: System domains

5.2 Prototyping

Since the scope of this work does not allow to realize all components and sequences of the credential ecosystem described in Section 4.4.1 and Section 4.4.2, fundamental elements have been implemented in the scope of proof of work. The aim is to implement basic features and considerations to build up the basis of the concept; whereby, the focus is set to back-end implementations. The prototyping was structured within three domains: a network domain, service provider domain and a user domain. Figure 5.1 shows an overview of the three domains. The network domain contains all functionalities within the Corda network, e.g. flows to create a DID document object. The provider domain covers all server-side implementations, which are not Corda specific. In conclusion, the user domain contains front-end and user-centric applications, such as the identity management tool (IMT).

The crucial element of the system is the decentralized public key infrastructure in the form of DID and DID document. In a first step, the capabilities of creating and updating DID documents as well as resolving a DID in the network were implemented. In a second step, the wallet and tools of the indy-SDK were integrated on the back-end. Moreover, the interface to front-end applications was developed. Finally, an example of an IMT was prototyped. It allows creating DID's, interacting with verifiable credential issuers as well as present credentials to a verifier.

5.2.1 Network Domain

In the network domain the feature of creating, updating DID documents, and resolving a DID for a DID document was added. Each DID document is stored in the form of a shared object in the network. It requires following main steps to extend the capability of creating a DID document object:

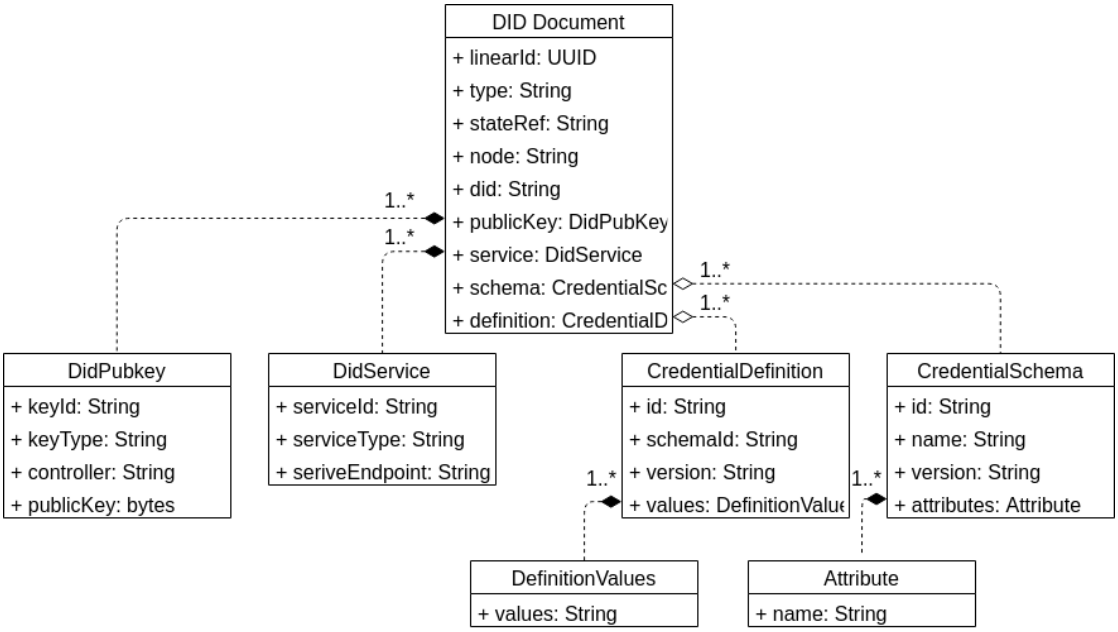

Figure 5.2: DID document data model

1. creating DID state object; specify the data model
2. creating DID document entity
3. realizing SQL schema
4. implementing Corda flow to create the DID document object

All DID documents are always shared between the creator node and the notary. Figure 5.2 illustrates the implemented DID document data model. A document can contain several public keys in the form of a DidPubKey object. Each public key an be used for authentication, encryption or other purposes. Services are defined in a DidService object. Each service object covers an id, a type and an endpoint, which should be an URL of the service. Furthermore, credential schemas and credential definitions are part of the document. These objects are required for verifiable credentials of the indy-SDK. A credential definition contains public cryptomaterial required for the camenisch-lysyanskaya signature of the zero-knowledge capable credentials. And, credential schema specifies the data model of a particular credential type. Also, the document consists of elements necessary for Corda transactions such as the linearId and stateRef. In Listing 5.1 the implementation of the model is shown.

Besides, a DID document update flow was implemented. It allows altering public keys, services, schemas and definitions. Furthermore, a resolver flow enables to request each DID document from the network. DID documents are always shared with the notary. Thus, the resolver flow performs a request from the notary.

Listing 5.1: An example of a DID document in Cardossier

```
{
  "data" :{
    "id" :"did:sb4b:2mwGekXcMhXhoMkvcM4Bfb",
    "type" :"didDocument",
    "schema" :null,
    "node" :"O=localhost, L=Zurich, C=CH",
    "stateRef" :null,
    "service" :{
      "did:sb4b:2mwGekXcMhXhoMkvcM4Bfb;inbox" :{
        "serviceId" :"did:sb4b:2mwGekXcMhXhoMkvcM4Bfb;inbox",
        "serviceType" :"CredentialInbox",
        "serviceEndpoint" :"O=localhost, L=Zurich, C=CH"
      }
    },
    "definition" :null,
    "publicKey" :{
      "did:sb4b:2mwGekXcMhXhoMkvcM4Bfb#keys-1" :{
        "keyId" :"did:sb4b:2mwGekXcMhXhoMkvcM4Bfb#keys-1",
        "keyType" :"Ed25519VerificationKey2018",
        "controller" :"did:sb4b:2mwGekXcMhXhoMkvcM4Bfb",
        "publicKeyBase58" :"eTZ6enk2YzQ3eTVoRG..",
      }
    },
  }
}
```

5.2.2 Provider Domain

The Java wrapper of the Indy-SDK is integrated and extended with more convenient data object. The SDK contains all wallet functionalities, see Section 5.1. A WalletService object is implemented, which defines a wallet object in the system and covers all the required methods. Multiple wallets can be created, each encrypted with a wallet pass-phrase. On a higher-level, the wallet is connected with the DID-document-create and DID-document-update flow. Furthermore, a REST API is written such that the wallets can be accessed from a front-end application, e.g. from the issuer.

An issuer service is implemented. We cover the described issuer in Section 4.4.1. Basic building blocks were realized: the wallet, issuer service API, network adapter for DID documents and partial implementation of the logic layer. Furthermore, the DID-authentication sequence (Figure 4.14) was added to the implementation and the IMT-to-API sequence flow of Figure 4.13 was realized. The issuer service was equipped with two demo credentials, a "name" credential and an "insurance" credential.

A prototype of the verifier was realized by implementing the basic building blocks: network adapted for DID documents and verification tools. The logic layer was integrated, and

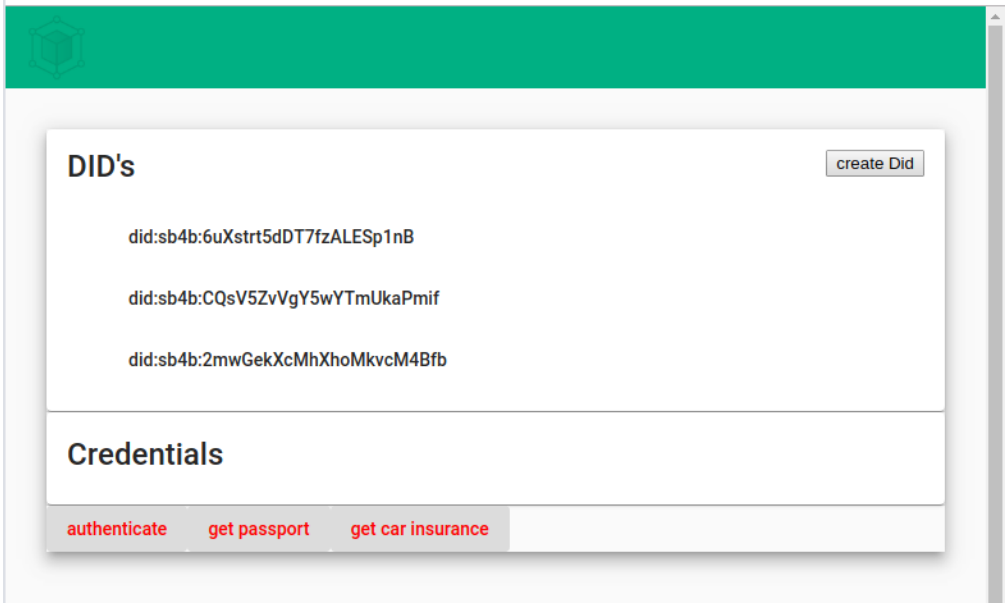

Figure 5.3: Home screen with multiple DID's

the interface was realized. It enables the interaction sequence of Figure 4.13 without credential revocation and trust policy.

5.2.3 User Domain

A demo identity management tool was developed to showcase the basic functionalities. The web application is making use of a wallet service from the back-end.

A first basic feature is to create and control DID's. An identity owner can create multiple ones, see Figure 5.3. It shows the home screen with multiple DID's in possession. The home screen lists up all DID's and credential of the identity owner. New DID's can be created with the button on the top right, which will create a new DID document on the ledger.

The identity owner can request credential from an issuer. Clicking on the button get a passport, he can select which identifier (DID) he is going to use for the connection, see Figure 5.4. Afterwards, a DID-authentication takes place in the background. When successfully authenticated (the identity owner can prove ownership of the DID) the issuer creates a new verifiable credential and transmits it to the identity owner. The credential is visualized on the home screen, see Figure 5.5. It contains the three claims "surname: Glauser", "first name: Remo" and "level of assurance: 1" in our example.

An identity owner can make use of his credentials by demonstrating them to a verifier. The integration between the verifier and the identity owner is implemented. An identity owner requests for service, whereby the verifier response with a request for a verifiable presentation. In our example, the verifier asks for the first name and surname of the identity

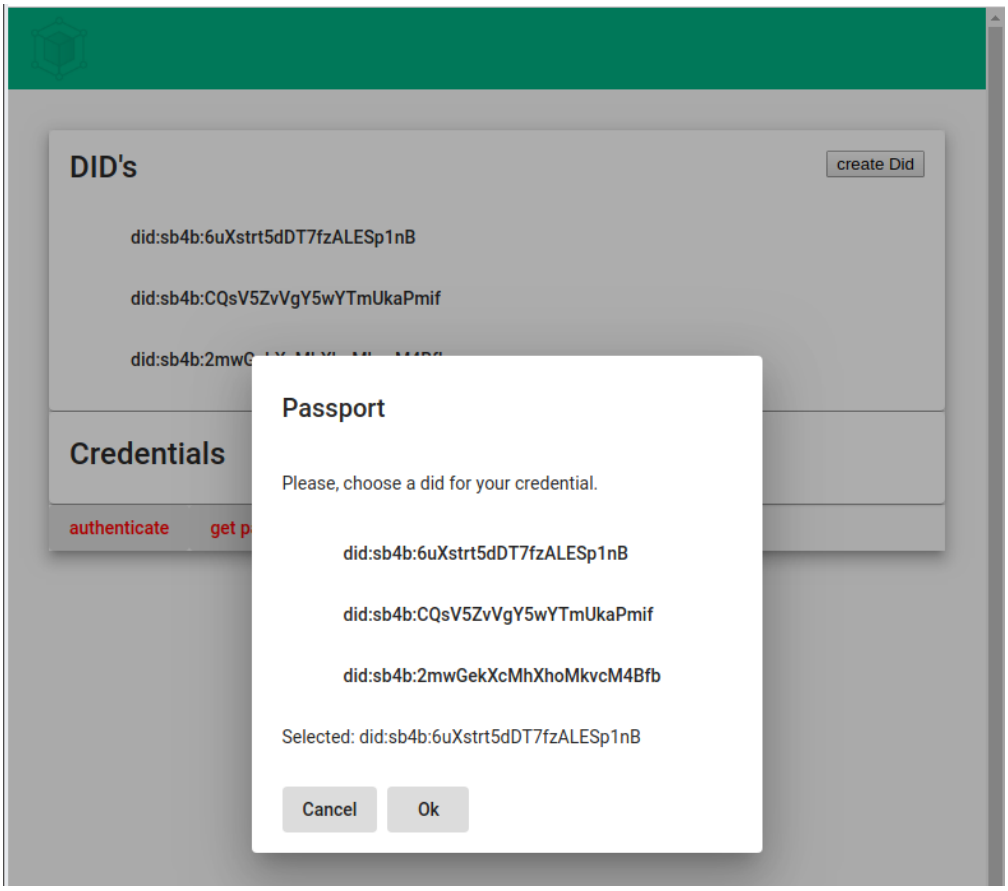

Figure 5.4: DID selection for before credential issuance

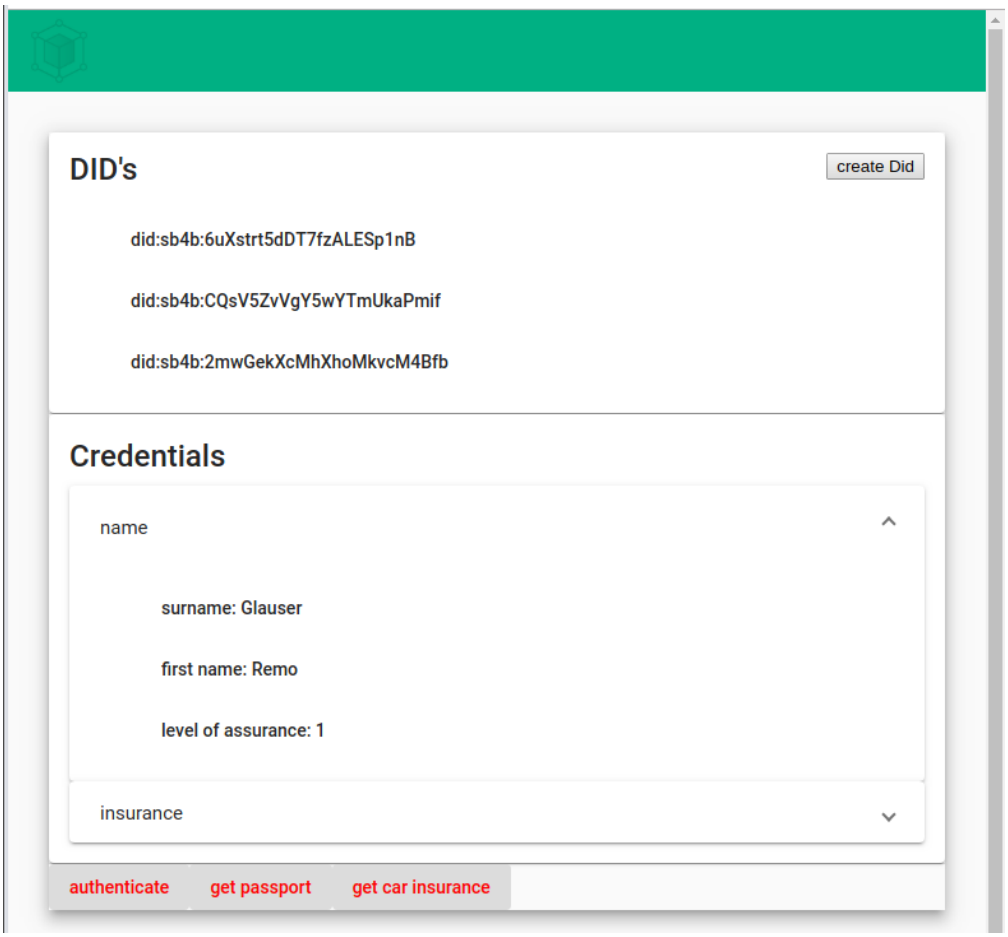

Figure 5.5: A new credential have been issued

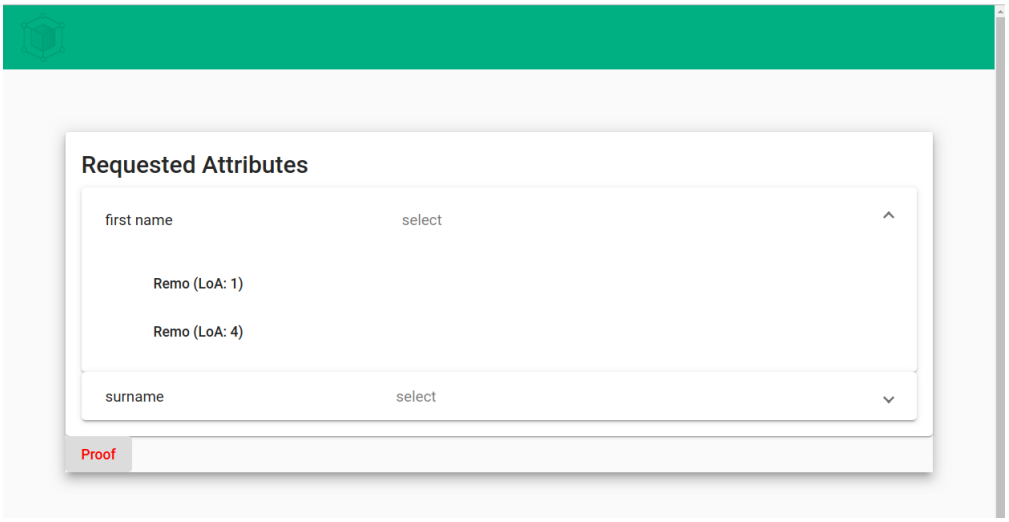

Figure 5.6: Selecting claims for a verifiable presentation

owner, see Figure 5.6. It shows how the user can select suitable claims for the request. In our case, there are two claims with the attribute "first name". One is asserting "first name: Remo" with a level of assurance (LoA) equals to 1 and the second with a LoA equals to 4. The claim with LoA 4 should be selected since the confidence of the issuer that the claim is valid is higher. Afterwards, all credential schemas and definition required for the proof are collected from the network. Finally, the verifiable presentation is created and transmitted to the verifier, which verifies the correctness of the presentation.

CHAPTER 6

Conclusion

We asked the question of how can users take control of their identity and interact with the Cardossier ecosystem with privacy and trust. For answering this question, system requirements on identity management were elaborated, and the suitability of self-sovereign identities was analyzed. Data privacy is one of the primary concern of Cardossier, especially when there are personal data involved. We have seen that the concept of self-sovereign identity removes the trusted third-party and gives control of the identity to the user. On its basis, it has a decentralized public key infrastructure. Privacy can be preserved on a degree not reached by other identity management approaches. Furthermore, SSI enables data interoperability through verifiable credentials. We investigate existing data models and implementations of SSI and present a concept best-suitable for Cardossier. The integration of the Hyperledger Indy-SDK was proposed. It enables minimal disclosure of personal information through zero-knowledge proofs, and it allows separating claims from identifiers. We discuss how these features can be utilized to prevent Identity correlation in Cardossier. Moreover, we present a wallet backup & recovery mechanism which does not rely on a trusted third-party or the participation of friends or family. Instead, trust is distributed among several nodes with direct authentication via an additional channel such as username/password or FIDO authenticators. We discuss the utilization of SSI in Cardossier for personal data, but also how it can be used to give user consent to Corda transaction as well as how the management of car ownership can benefit. Besides, a system architecture of the credential ecosystem is presented. We define for roles: a credential issuer, an agent, an identity owner and a credential verifier; and we elaborate their individual software architecture and interaction sequences. Finally, crucial elements of the architecture were implemented in the scope of a proof-of-work.

In conclusion, we demonstrated how to provide, and partially implement as proof-of-work, an identity management that preserves privacy of end-users in decentralized ecosystems like Cardossier.

The technology of self-sovereign identity is just on the rise and not yet widely adopted in production. Tools and libraries are rapidly developed further, and so experience the used Indy-SDK. When making the proposed concept productive in Cardossier, this fact should be considered by following on the emerging W3C community standards. Since user-centric identity management requires the active participation of the user - users must agree in digital identity/passport, download an app, request for credentials, etc. - a self-sovereign system

can only slowly be introduced. It requests for a hybrid system between federated and self-sovereign identities. Organization participating in Cardossier should administrate the user's identities and offer to the users to receive control about their identities whenever they desire. Further investigation of specifying this process might be considered. Besides, the usability of the identity management tool for the end-user is crucial to get their acceptance for the application. Therefore, a great effort to reach the best user experience should be taken. Moreover, only selective disclosure of car events is implemented. Zero-knowledge capable events might be a desired property for minimizing the disclosure in a car data market. Cardossier is based on a permissioned distributed ledger, so is the presented solution. One aspect of the principles of SSI is transparency: The systems used to administer and operate a network of identities must be open, both in how they function and in how they are managed and updated. Permissioned networks do not entirely offer it since network participation is regulated, however, permssionless blockchains lake on transaction scalability. A promising solution to solve the issue for Decentralized Identities are sidetrees, which were presented by the Decentralized Identity Foundation. Further investigation on its suitability for Cardossier might be performed.

Bibliography

[1] J. Camenisch and E. Van Herreweghen, "Design and implementation of the idemix anonymous credential system", in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, ser. CCS '02, Washington, DC, USA: ACM, 2002, pp. 21–30, ISBN: 1-58113-612-9. DOI: 10.1145/586110.586114. [Online]. Available: http://doi.acm.org/10.1145/586110.586114.

[2] C. Adams and S. Lloyd, *Understanding PKI: concepts, standards, and deployment considerations*. Addison-Wesley Professional, 2003.

[3] R. Oppliger, "Microsoft .net passport and identity management", *Information Security Technical Report*, vol. 9, no. 1, pp. 26–34, 2004, ISSN: 1363-4127. DOI: https://doi.org/10.1016/S1363-4127(04)00013-5.

[4] A. Jøsang and S. Pope, "User centric identity management", in *AusCERT Asia Pacific Information Technology Security Conference*, Citeseer, 2005, p. 77.

[5] L. Benedictus, "A brief history of the passport", *The Guardian*, Nov. 2006, ISSN: 0261-3077. [Online]. Available: https://www.theguardian.com/travel/2006/nov/17/travelnews (visited on 04/04/2019).

[6] T. Dierks and E. Rescorla, "The transport layer security (tls) protocol version 1.2", Internet Engineering Task Force (IETF), California, USA, Tech. Rep., 2008.

[7] D. W. Chadwick, "Federated identity management", in *Foundations of Security Analysis and Design V: FOSAD 2007/2008/2009 Tutorial Lectures*, A. Aldini, G. Barthe, and R. Gorrieri, Eds. Springer Berlin Heidelberg, 2009, pp. 96–120, ISBN: 978-3-642-03829-7. DOI: 10.1007/978-3-642-03829-7_3. [Online]. Available: https://darticleoi.org/10.1007/978-3-642-03829-7_3.

[8] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures", in *High-speed high-security signatures*, vol. 2, Springer, 2012, pp. 77–89.

[9] P.-L. Aublin, S. B. Mokhtar, and V. Quéma, "Rbft: Redundant byzantine fault tolerance", in *2013 IEEE 33rd International Conference on Distributed Computing Systems*, IEEE, 2013, pp. 297–306.

[10] C. Paquin, "U-prove technology overview", *Microsoft Corporation*, p. 23, 2013.

[11] M. Sporny, D. Longley, G. Kellogg, M. Lanthaler, and N. Lindström, "Json-ld 1.0", W3C Recommendation, Tech. Rep., 2014. [Online]. Available: https://www.w3.org/TR/json-ld.

[12] "Decentralized public key infrastructure", 2015, [accessed 15-April-2019]. [Online]. Available: https://danubetech.com/download/dpki.pdf.

[13] C. Allen. (Apr. 2016). The path to self-sovereign identity, [Online]. Available: http://www.lifewithalacrity.com/2016/04/the-path-to-self-soverereign-identity.html (visited on 04/04/2019).

[14] M. Crosby, P. Pattanayak, S. Verma, V. Kalyanaraman, *et al.*, "Blockchain technology: Beyond bitcoin", *Applied Innovation*, vol. 2, no. 6-10, p. 71, 2016.

[15] R. Gendal Brown, J. Carlyle, I. Grigg, and M. Hearn, "Corda: An introduction", *R3 CEV*, Sep. 2016. DOI: 10.13140/RG.2.2.30487.37284.

[16] M. M. L. L. Initiative, *Blockcerts-an open infrastructure for academic credentials on the blockchain*, Medium, Oct. 2016. [Online]. Available: https://medium.com/mit-media-lab/blockcerts-an-open-infrastructure-for-academic-credentials-on-the-blockchain-899a6b880b2f.

[17] D. Reed, J. Law, and D. Hardman, "The technical foundations of sovrin", in *The Technical Foundations of Sovrin*, Sep. 2016.

[18] V. Morabito, "Business innovation through blockchain", *Cham: Springer International Publishing*, 2017.

[19] M. Sabadello, *A universal resolver for self-sovereign identifiers*, Medium, Nov. 2017. [Online]. Available: https://medium.com/decentralized-identity/a-universal-resolver-for-self-sovereign-identifiers-48e6b4a5cc3c.

[20] C. Fei, J. Lohkamp, E. Rusu, K. Szawan, K. Wagner, and N. Wittenberger, "Self-sovereign and decentralised identity by design", Mar. 2018.

[21] C. Jagers, *The new blockcerts mobile app*, Medium, Apr. 2018. [Online]. Available: https://medium.com/learning-machine-blog/the-new-blockcerts-mobile-app-eea18053f526.

[22] M. Loepfe, "Wie blockchain vertrauen und digitalisierung fördert", p. 6, 2018.

[23] "Microsoft decentralized identity", 2018. [Online]. Available: https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RE2DjfY.

[24] "Sovrin protocol and token white paper", Jan. 2018. [Online]. Available: https://sovrin.org/wp-content/uploads/Sovrin-Protocol-and-Token-White-Paper.pdf.

[25] A. Young and S. Verhulst, "Self sovereign identity for government services in zug, switzerland", p. 11, Oct. 2018.

[26] I. Bauer, L. Zavolokina, F. Leisibach, and G. Schwabe, "Exploring blockchain value creation: The case of the car ecosystem", in *52nd Hawaii International Conference on System Sciences*, 2019, p. 10.

[27] D. Buchner and A. Simons, *Toward scalable decentralized identifier systems*, May 2019. [Online]. Available: https://techcommunity.microsoft.com/t5/Azure-Active-Directory-Identity/Toward-scalable-decentralized-identifier-systems/ba-p/560168.

[28] "Credential — Wikipedia, the free encyclopedia", *Wikipedia*, May 2019, [accessed 8-May-2019]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Credential&oldid=895436374.

[29] *Did resolver for https domains*, GitHub repository, [accessed 15-May-2019], Apr. 2019. [Online]. Available: https://github.com/uport-project/https-did-resolver.

[30] *Indy-sdk: Api*, GitHub Repository, [commit: 2977b26b], May 2019. [Online]. Available: https://github.com/hyperledger/indy-sdk/blob/master/libindy/src/api.

[31] *Indy-sdk: How credential revocation works*, GitHub Repository, [commit: 4c0efa5e], May 2019. [Online]. Available: https://github.com/hyperledger/indy-sdk/blob/master/docs/concepts/revocation/cred-revocation.md.

[32] *Notary demo*, GitHub Repository, [commit: 0fdb2674], May 2019. [Online]. Available: https://github.com/corda/corda/tree/master/samples/notary-demo.

[33] D. Reed, M. Sporny, D. Longley, C. Allen, R. Grant, and M. Sabadello, "Decentralized identifiers (dids) v0.12", W3C Community Group, Tech. Rep., 2019. [Online]. Available: https://w3c-ccg.github.io/did-spec.

[34] "Sidetree protocol specification", Tech. Rep., Jun. 2019, [git commit: 2d7a5f29]. [Online]. Available: https://github.com/decentralized-identity/sidetree/blob/master/docs/protocol.md.

[35] M. Sporny, D. Longley, and D. Chadwick, "Verifiable credentials data model 1.0", W3C Community Group, Tech. Rep., 2019. [Online]. Available: https://www.w3.org/TR/verifiable-claims-data-model.

[36] "Storage and compute nodes for decentralized identity data and interactions", May 2019. [Online]. Available: https://github.com/decentralized-identity/identity-hub.

[37] "Verkauf von neuwagen sinkt in der schweiz erneut", *Tages-Anzeiger*, Jan. 2019, ISSN: 1422-9994. [Online]. Available: https://www.tagesanzeiger.ch/wirtschaft/standardverkauf-von-neuwagen-sinkt-in-der-schweiz-erneut/story/22560129.

[38] "Web authentication: An api for accessing public key credentials level 1", W3C Recommendation, Tech. Rep., Mar. 2019. [Online]. Available: https://www.w3.org/TR/webauthn.

[39] Wikipedia contributors, "Federated identity — Wikipedia, the free encyclopedia", 2019, [accessed 6-April-2019]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Federated_identity&oldid=884016571.

[40] ——, "Identity (social science) — Wikipedia, the free encyclopedia", 2019, [accessed 6-April-2019]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Identity_(social_science)&oldid=886772120.

[41] *A new approach to digital identity*, accessed 19 May 2019. [Online]. Available: https://didproject.azurewebsites.net/.

[42] "Auto-schweiz: Statistiken", [Online]. Available: https://www.auto.swiss/statistiken.

[43] *Cardossier - managing the life cycle of a car with blockchain technology.* [Online]. Available: https://cardossier.ch/ (visited on 05/10/2019).

[44] *Carfax*. [Online]. Available: https://www.carfax.eu.

[45] "Corda, designed for business", R3 LLC, New York City, USA, Report. [Online]. Available: https://www.corda.net/discover/technology.html (visited on 02/06/2019).

[46] *Fido2*, [accessed 31 May 2019]. [Online]. Available: https://fidoalliance.org/fido2/.

[47] M. Hearn, "Corda: A distributed ledger", p. 56, [Online]. Available: https://docs.corda.net/_static/corda-technical-whitepaper.pdf.

[48] *Hyperledger indy*, [accessed 11-May-2019]. [Online]. Available: https://www.hyperledger.org/projects/hyperledger-indy.

[49] *Identity hub javascript sdk*, [accessed 18 May 2019]. [Online]. Available: https://identity.foundation/hub-sdk-js/.

[50] *Identity mixer | anonymous credentials for strong accountability and privacy.* [accessed 11-May-2019]. [Online]. Available: https://idemix.wordpress.com.

[51] D. C. Lundkvist, R. Heck, J. Torstensson, Z. Mitton, and M. Sena, "Uport: A platform for self-sovereign identity", p. 17,

[52] B. Pelle and T. Joel, *Erc: Lightweight identity*, GitHub, [accessed 15-May-2019]. [Online]. Available: https://github.com/ethereum/EIPs/issues/1056.

[53] S. Peyrott, *Introduction to web authentication: The new w3c spec*, [accessed 8 June 2019]. [Online]. Available: https://auth0.com/blog/introduction-to-web-authentication.

[54] *Sovrin foundation*, [accessed 11-May-2019]. [Online]. Available: https://sovrin.org.

[55] *Transaction fueling server*, uPort, [accessed 15-May-2019]. [Online]. Available: https://developer.uport.me/rest-apis/fuel-server.

[56] *Uport credentials*, uPort, [accessed 15-May-2019]. [Online]. Available: https://developer.uport.me/uport-credentials/login.

[57] *Uport developer portal*, uPort, [accessed 15-May-2019]. [Online]. Available: https://developer.uport.me.

[58] *Uport protocol specs*, GitHub repository, [accessed 15-May-2019]. [Online]. Available: https://github.com/uport-project/specs.

[59] *Verifiable organizations network*, [accessed 11-May-2019]. [Online]. Available: https://vonx.io/.

[60] "Iso/iec 24760-1:2011: A framework for identity management – part 1: Terminology and concepts", International Organization for Standardization, Geneva, CH, Dec. 2011. [Online]. Available: http://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/05/79/57914.html.