



Emanuel Kirchengast, BSc

Qualified eID derivation to Self-Sovereign Identities

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

O.Univ.-Prof., Dipl-Ing., Dr.techn. Reinhard Posch

Institute of Applied Information Processing and Communications (IAIK)

Dipl-Ing. Andreas Abraham

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date

Signature

Acknowledge

I want to thank Andreas Abraham for his support and supervision during the development of the concept, the implementation, and the writing of this thesis. His feedback helped to shape this thesis to the piece I proudly present today.

Furthermore, I want to thank my friends and family for their support throughout my studies.

Emanuel Kirchengast
Graz, December 2018

Abstract

With the rise of digitalization, the requirements for identity management on the Internet grow. Especially with a rising number of sensitive services, the need for access to flexible and trustworthy electronic identities (eID) increases. However, in traditional identity management systems, Identity Providers (IdP) orchestrate identities. They often control identity data, restrict identity domains and monitor the services a user accesses. The Self-Sovereign-Identity (SSI) concept tackles these problems by empowering users to store personal information locally and to authenticate to services without the use of a trusted third party. A network of nodes operating a distributed ledger system guarantees the trust within the system.

Nevertheless, the migration from existing identity management (IdM) systems to SSI systems presents a source of errors, due to the SSI's architecture that does not involve an IdP. Therefore, this thesis presents a concept that follows an identity derivation process without a trusted third party. Instead of being entirely migrated into a new architecture, an SSI system may import qualified eIDs from a traditional IdM system without dropping the level of assurance. The existing systems remain unaltered due to the introduction of a broker called the identity agent. The agent coordinates the derivation and revocation process. Since the agent does not reach the same level of assurance as a qualified IdP, a decentralized network of nodes in the SSI system executes a consensus protocol to preserve the trust and ensures the eIDs authenticity.

This thesis demonstrates the concept's feasibility with the implementation of an eID derivation from the eIDAS network to the Sovrin SSI system. Further, a demonstrator wraps the implementation into a user-friendly web application, and a security evaluation guarantees the soundness of the proposed derivation process. These findings provide a starting point for further research on the relatively new domain of SSI systems.

Kurzfassung

Bedingt durch die zunehmende Digitalisierung wächst auch die Bedeutung für Identitätsmanagement (IdM) im Internet. Mit dem Zunehmen von elektronischen Dienstleistungen mit sensiblen Benutzerdaten, steigt auch die Nachfrage nach vertrauenswürdigen elektronischen Identifikationsinformationen. Dennoch vertrauen viele traditionelle Identitätsmanagementbetreiber der Nutzung von Identitätsanbietern. Diese können in den meisten Fällen auf gespeicherte Nutzerdaten zugreifen und kontrollieren die Services, die der Benutzer erreichen kann. Diese Probleme werden von selbstsouveränen Identitätssystemen (SSI) durch lokale Speicherung und Verwaltung der persönlichen Daten gelöst. Ein Netzwerk aus Knoten bildet die Vertrauensbasis innerhalb des Systems und speichert gemeinsam getroffene Entscheidungen auf verteilten Kassenbüchern. Da SSI-Systeme keine Identitätsanbieter benötigen, kann die Migration von einem bestehenden IdM-System zu einem SSI-System zu Komplikationen führen. Aus diesem Grund liegt der Fokus dieser Arbeit auf der Ableitung von elektronischen Identitäten, anstelle einer Migration zu einem SSI-System werden nur die Identitätsdaten von einem existierenden IdM-System abgeleitet. Dadurch kann ein Benutzer von den Vorzügen einer SSI profitieren. Die bestehenden Systeme müssen für die Ableitung nicht verändert werden, stattdessen wird in vorliegender Arbeit ein Identitätsvermittler als Schnittstelle vorgestellt. Der Vermittler koordiniert den Widerrufungs- sowie den Ableitungsprozess von Identitätsinformationen, der ohne vertrauenswürdige Dritte auskommt. Stattdessen garantiert ein Konsensprotokoll die Authentizität der abgeleiteten Daten. Die Machbarkeit des in der Arbeit präsentierten Konzepts wird durch eine Implementierung und einen Demonstrator untermauert. Die Implementierung leitet Daten aus dem eIDAS-Netzwerk ab und verwendet Sovrin als SSI-System. Des Weiteren beschäftigt sich folgende Arbeit mit der Sicherheit des beschriebenen Konzepts, die in einer eigenen Sicherheitsanalyse bestätigt wird. Die gewonnenen Erkenntnisse sollen als Ausgangspunkt für weitere Forschung im noch jungen Gebiet der SSI-Systeme dienen.

Contents

1. Introduction	1
1.1. Methodology	2
1.2. Contributions	3
1.3. Outline	3
2. Preliminaries	4
2.1. eIDAS	4
2.1.1. eIDAS Architecture	4
2.1.2. Qualified Electronic Identity	5
2.1.3. Security Assertion Markup Language (SAML)	6
2.1.4. Mobile Phone Signature	7
2.2. Decentralized Identities	8
2.3. Verifiable Claims	10
2.3.1. Structure of Verifiable Claims	10
2.3.2. Verifiable-Claim-Ecosystem	11
2.3.3. Data Model	12
2.4. Consensus Protocol	12
2.4.1. Byzantine Fault Tolerance	13
2.4.2. Practical Byzantine Fault Tolerance Protocol	14
2.4.3. Redundant Byzantine Fault Tolerance Protocol	16
2.4.4. Indy Plenum	17
2.5. Self Sovereign Identities	17
2.6. Hyperledger Indy	18
2.6.1. Layers of Indy	19
2.6.2. Technical Functionality	22
2.6.3. Indy Workflow	24
2.6.4. Codebase	28
2.6.5. Multi-signatures	28
3. Related Work	29
3.1. Identity Systems	29
3.1.1. ShoCard	29
3.1.2. uPort	30

Contents

3.2. Sovrin Integration	31
3.3. Identity Derivation	31
4. Concept	33
4.1. Concept Architecture	33
4.2. Derivation	34
4.3. Revocation	36
5. Implementation	38
5.1. Client Agent	39
5.2. eIDAS Agent	41
5.3. Validator Nodes	45
6. Demonstrator	48
7. Discussion	52
7.1. Properties	52
7.2. Design Decisions	53
7.2.1. Assertion Inside the Claim	53
7.2.2. Identity Interface issues Claims	54
7.2.3. Individual Revocation	55
8. Security Analysis	56
8.1. Target of Evaluation and Actors	56
8.2. Assumptions	58
8.3. Assets	59
8.4. Threat Agents	60
8.5. Threats	61
8.6. Security Objectives	63
8.7. Countermeasures	66
8.8. Conclusion	67
9. Future work	68
10. Conclusion	70
Appendices	72
A. Listings	73
B. Screenshots	79

Contents

Bibliography

82

List of Figures

2.1.	An overview of the eIDAS architecture	5
2.2.	Sequence diagram of SAML assertion request/query profile	6
2.3.	Set of verifiable claims	10
2.4.	Relationship between roles in Verifiable-Claim-Ecosystem	11
2.5.	Generals' Problem with three loyal nodes	13
2.6.	Generals' Problem with two loyal lieutenants and a traitorous general	13
2.7.	Generals' Problem with one traitorous lieutenant	13
2.8.	Generals' Problem with four loyal nodes	14
2.9.	Generals' Problem with four nodes and one traitorous general	15
2.10.	Generals' Problem with four nodes and one traitorous lieutenant	15
2.11.	PBFT execution	16
2.12.	RBFT execution	17
2.13.	Visualization of the most relevant parties in Indy	20
2.14.	A new identity is added to Indy	24
2.15.	Bob establishes a communication channel with Alice	25
2.16.	An issuer creates Schema and Credential Definitions	26
2.17.	A Proofer requests a claim from an issuer	27
4.1.	Proposed architecture	34
4.2.	Sequence diagram showing the identity transformation process	35
4.3.	Sequence diagram showing the concept for revocation	37
5.1.	Components in concept and PoC	38
6.1.	Select authentication method	48
6.2.	Authentication field	49
6.3.	Field to insert TAN from a short message	49
6.4.	Summary after claim creation is complete	50
8.1.	Identity agent, SSI nodes and the ledger are part of the TOE	57
B.1.	The demonstrator's starting page	79
B.2.	SAML request creation	80
B.3.	The response from the eIDAS node	81

1. Introduction

“Who are you?” - This regularly used phrase in the English language usually requests for an unfolding of the identity. Familiar with the process of identification and authentication, we show passports when crossing borders, discount cards when going shopping and membership-cards before entering a gym without overthinking. Everyday life in modern society is full of situations where the provision of identity plays an important role and schemes for registration and verification of identities are ubiquitous. Due to digitalization and the shift of services from the real world to the virtual space, the need for electronic IDs (eID) rises. According to European law [1], an eID contains identity information that could provide authentication and identification for other services. Banks, insurances or eGovernments can improve their services if an eID links to a real-world identity.

Even though identification and authentication became crucial components of the web, the fundamental structure of the Internet does not natively support these processes, or like Kim Cameron, Chief Architect of Identity for Microsoft once said: “The Internet was created without an identity layer.” [2]. Due to the lack of native integration, several identity models have been developed that Bertino and Takahashi [3] describe in higher detail. The models depict the relationship between the subject, which represents an identity holder, the entity that controls the user’s data, called an Identity Provider (IdP) and the Service Provider (SP), who offers a service to the subject [4]. The isolated model represents the most basic model and describes an SP that operates an IdP. This model challenges identity owners because they need to register a new digital identity for each service. More convenient is the central model, where the IdP and the SP run separately from each other. After a subject’s registration at an IdP, the same identity can be used for authentication with several SPs. One prominent example of a central identity model represents the single sign-on (SSO). A more advanced approach represents the federated identity model, which connects distributed IdPs and allows a subject’s authentication across domains. All of these models face the challenge of securing the personal data in the IdPs but with a growing amount of information the honeypot for attackers increases. Therefore, within the last years, attackers compromised billions of

1. Introduction

profiles, including incidents of Equifax¹, Yahoo², and Facebook³ that reached mass-media-attention. In contrast, the user-centric model [5] aims for higher privacy and more control by storing personal data locally at the subject. Even though each model focuses on different designs, all of them depend on an IdP. A curious IdP could keep track of the services a subject connects to, which invades the individual's privacy. An even worse scenario poses a malicious IdP that could restrict the services a subject may connect. The introduction of the Self-Sovereign Identity (SSI) model solves this issue by enabling a subject to directly connect to an arbitrary service without the need for an IdP. Even though the SSI structure aims for more privacy, without eIDs and trusted IdPs sensitive services like banking could not be used. According to Ruff [6], the SSI model is still in an early stage. Since SSI systems do not need IdPs, full migrations from traditional identity management (IdM) systems where the IdPs play an essential role, can lead to erroneous behavior.

1.1. Methodology

In this thesis, we want to empower subjects to manage their credentials independently from an IdP and to prevent problems during the migration process from one identity model to the other. Therefore, we propose an import process that derives qualified eIDs from an existing trusted IdM system to an SSI system. This approach does not force the existing system to change its identity model. Nevertheless, a user could take advantage of the SSI properties like being entirely in charge of the identity data and using services without an IdP overseeing the process. Since this thesis describes an import process, the existing IdM system remains untouched and coexists with the SSI system, which omits a potentially erroneous migration process.

During the import process the data format changes, which drops the level of assurance of the eID. To circumvent this, we designed a process that preserves the eID's authenticity and trust. Following the motive for decentralization in SSI systems, our concept for the transformation does not need a trusted third-party. Instead, we introduce an approach that utilizes the distributed trust of the SSI network for accomplishing the eID transformation in the derivation process.

¹www.zdnet.com/article/hackers-stole-more-equifax-data-than-first-thought/.

²money.cnn.com/2017/10/03/technology/business/yahoo-breach-3-billion-accounts/.

³www.abc.net.au/news/2018-09-29/50-million-facebook-profiles-hacked/10319886.

1. Introduction

1.2. Contributions

The thesis includes the following four contributions:

1. **The Concept for the derivation process:** In our concept, we propose a method that derives a subject's attributes from an existing IdM system to an SSI system without making use of a trusted third party. We describe the architecture and elaborate the existing components and the ones we newly introduce. The thesis further contains a detailed insight into the data flow for identity derivation and revocation.
2. **Implementation of a Proof-of-Concept (PoC):** Based on the concept we create a PoC which imports qualified data and transforms the data format. One key requirement demands the existing systems to remain unchanged. Therefore, we create an interface that handles the network traffic and overcomes message format barriers. We further implement the transformation process that guarantees the level of assurance of the derived eID.
3. **Demonstrator of PoC:** We provide an easy-to-use demonstrator, which runs in a publicly available web application. An external user can test our implementation and get an impression of how the identity derivation process looks and works.
4. **Security Evaluation:** This thesis features a security evaluation based on the Common Criteria [7] process. It introduces actors, assets, attackers and threats. Subsequently, the analysis evaluates the mitigation mechanisms in concept and implementation that protect the assets against attackers.

1.3. Outline

This thesis presents our findings. Chapter 2 provides information about technologies that we apply in this project. Later, chapter 3 describes related systems, concepts, and research. Afterward, we introduce the concept for the identity derivation in chapter 4. Next, chapter 5 gives a detailed insight into the PoC's implementation. After that, in chapter 6, we describe the demonstrator from an end user's perspective. Subsequently, chapter 7 discusses design decisions from the concept and implementation. Chapter 8 deals with the security analysis, one of the main contributions of this thesis. Finally, chapter 9 gives an outlook for future research and chapter 10 finishes this thesis with a conclusion.

2. Preliminaries

Since we want to import data from a traditional identity management (IdM) system into a decentralized identity system, we utilize several existing technologies. Therefore, the following chapter gives an insight into the most relevant technologies and frameworks that we use in this thesis. The chapter starts with an introduction to the eIDAS network that is followed by a description of the concepts of DIDs and Verifiable Claims. In the end, we focus on the Byzantine fault tolerance protocol and the decentralized identity framework Hyperledger Indy.

2.1. eIDAS

The electronic identification, authentication and trust services (eIDAS) [8] regulation of the European Union builds the legal basis for an interoperability network between the member states. The regulation aims to connect the existing national systems to make them interoperable by specifying the interfaces between them. The regulation offers opportunities for member states to catalyze the digitalization of national eGovernment strategies. It could facilitate the own citizens and businesses along with the ones from other member states to reach these national resources.

The following section describes the architecture of the eIDAS network [9] [10] and the concept of qualified eIDs. These parts succeed in a description of the eIDAS's SAML protocol and an introduction to the mobile phone signature, which represents the Austrian eID authentication solution.

2.1.1. eIDAS Architecture

As mentioned before, eIDAS builds on the national eID solution of each member state. Therefore a guideline [9] from the European Union specifies an architecture for adding interoperability between different states. The guideline states that each member state needs to implement an "eIDAS node" [10], which consists of an "eIDAS connector" and

2. Preliminaries

an “eIDAS services”. Those two sub-components handle the requests and responses from and to the relying parties and the national Identity Providers (IdP). The eIDAS nodes are connected and therefore, provide a network between the member states. Figure 2.1 depicts these connections. If a citizen of country A wants to use an application connected to an eIDAS node of country B, the node redirects the authentication request to the eIDAS node of country A. This node redirects the request to the national IdP, where the citizen can use the login method of the home country for authentication.

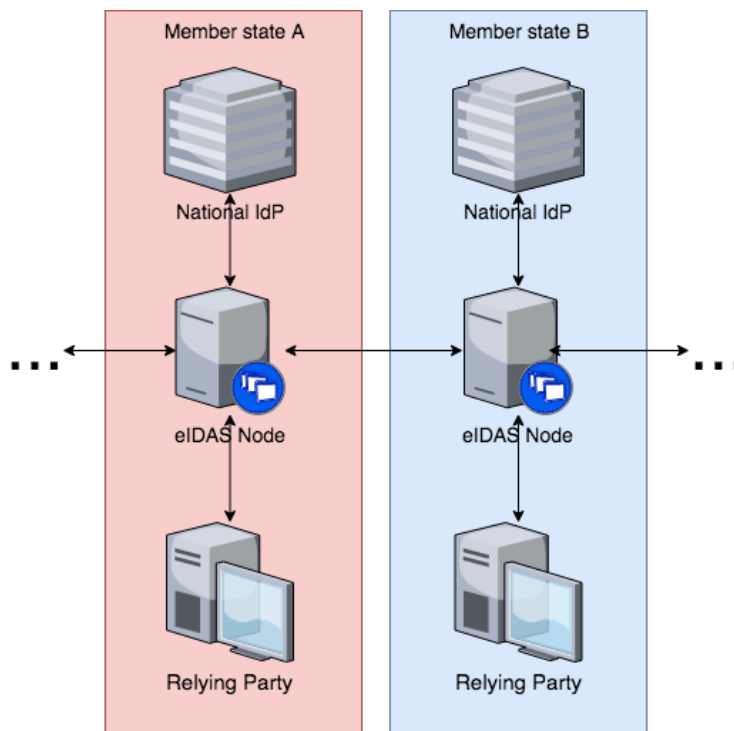


Figure 2.1.: An overview of the eIDAS architecture

2.1.2. Qualified Electronic Identity

An eID uniquely identifies a person in a specific domain, which increases the trust between two parties during electronic communication. Therefore, eIDs are utilized in applications dealing with sensitive data. Furthermore, an eID must implement functions for identification, signature creation, and authentication.

2. Preliminaries

Additionally to the properties stated before, a qualified eID ensures that an authority bails for the correctness of the attributes. Therefore, an identity owner must follow a strong authentication process to verify the identity. Due to this, qualified eIDs are trustworthily treated and recognized by several services including eGovernment applications.

2.1.3. Security Assertion Markup Language (SAML)

The Security Assertion Markup Language (SAML) [11] exchanges authentication and authorization data between connected parties. The XML protocol runs across domains and allows single-sign-on (SSO) and other capabilities of federated identity structure (see chapter 1). The so-called SAML authorities describe domains that individually consist of IdP and SP. While the IdP stores the user's data and provides an authentication mechanism, the SP offers a service to the user. The separation of the authentication from the actual service advantageously allows one IdP to serve numerous SPs. Instead of multiple implementations, only the IdP has to ensure a secure authentication method and credential storage. In the case of the eIDAS architecture, SAML version 2.0 allows the eIDAS nodes and the connected services to transmit personal data while maintaining trust within the network [10].

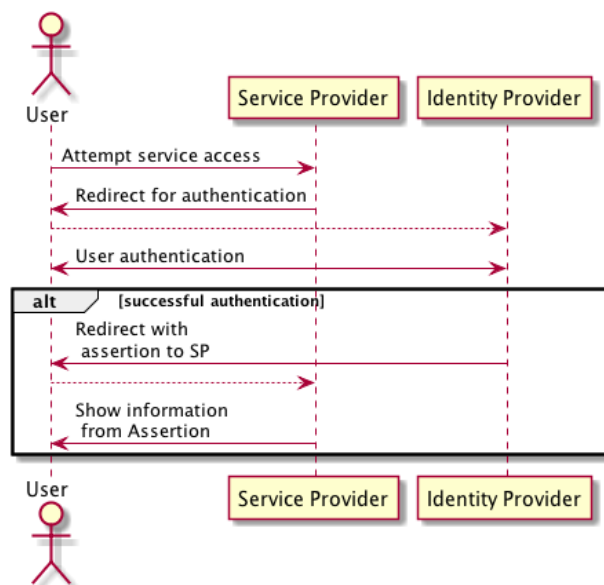


Figure 2.2.: Sequence diagram of SAML assertion request/query profile

2. Preliminaries

Besides authentication, SAML provides so-called “SAML Assertions”. These XML messages contain a subject’s information and a signature guaranteeing authenticity. In this thesis, we use assertions in the context of the “SAML Assertion Query/Request Profile” [12]. This schema defines a procedure, where an SP requests a subject’s data from an IdP. The process depicted in Figure 2.2 starts with a user consuming an SP’s service that requires personal information. The SP creates an assertion request, which contains a list of attributes it wants to query from the IdP. Next, the SP redirects the user with the request to the IdP. Once a user completes the authentication, the IdP creates a cryptographically verifiable assertion that includes the requested attributes of the user. The user is redirected to the SP, who decrypts the assertion, verifies the signature and may subsequently use the trusted attributes in its service.

2.1.4. Mobile Phone Signature

The mobile phone signature [13] (in German “Handysignatur”) depicts one authentication scheme for an Austrian national IdP. The method represents the digital equivalent of an identity card and employs signing and authentication mechanisms. Citizens may utilize the mobile phone signature to log in to governmental services to make their tax returns or request their criminal records. The mobile phone signature follows the legal requirements of the eIDAS regulations for interaction with qualified eIDs. As stated before, eIDs of this kind must have an authority, which vouches for them. For the Austrian mobile phone signature, the organization A-Trust provides this service. A-Trust supplies secure hardware-modules and credential storage to Austrian citizens in compliance with the eIDAS regulations [14]. Further, qualified eIDs require strong authentication, which provides the mobile phone signature with two-factor authentication.

In Austria, several possibilities exist how a resident can proof the identity [15]. After the verification of the real identity, the citizen specifies a password and the personal phone number. Subsequently, a resident can start to use the mobile phone signature. For this purpose, she visits a service which utilizes this authentication method and enters her phone number and the password. After successful verification, the second-factor-authentication comes into effect. If the citizen uses the mobile phone signature app¹, the browser shows a QR-code that must be scanned. Otherwise, the IdP sends a short message with a six-digit once-only-token to the resident’s phone. The citizen must enter the token into a text field in the browser to complete the second-factor-authentication-step. Subsequently, the user can access personal credentials which allow the creation of signatures or the authentication of services.

¹www.handy-signatur.at/mobile/TanAppUpgrade/.

2.2. Decentralized Identities

In current systems, identity-relevant information like identifiers, keys, and endpoints are often stored in centralized registers. In IdM systems, for example, identity-data are governed by IdPs and certificates are issued and revoked in Public Key Infrastructures (PKIs) by centralized Certificate Authorities (CA). Decentralized Identities (DIDs) were introduced to move the control over identities to the owner. The standard for DIDs [16] is currently developed in an open process by the W3C credentials community group². The standard is designed for interoperability and portability to make identities available on different systems and move them between systems. Even though [16] defines the core functionality of DIDs each implementation is obligated to create its own DID-Specification. The specification defines the create, read, update and delete (CRUD) behavior of the DIDs in the network. Each DID-Entry consists of two parts: a DID and a DID-Document. The DID is a URL pointing to the DID-Document, which contains the relevant metadata and the cryptographic keys.

The DID's structure is conformal to the RFC3986 [17] and consists of three parts separated by ":". First, "did", second, the "did-method" and third, an "idstring". The DID-method is specified in the DID-specification, therefore unique and should not be longer than five characters. It is called DID-Reference, if the symbols "/" or "#" are following a DID because they are indicating a "path" or "fragment". A DID-Reference is pointing to further resources like publicKeys. The listing 2.1 shows the structure and an example DID-URL.

```
DID Structure and Example:  
did = "did:" method ":" specific-idstring  
did:example:123456789abcdefghi  
  
DID-Reference Structure and Example:  
did-reference = did ["/" did-path] ["#" did-fragment]  
did:example:123456789abcdefghi#keys-1
```

Listing 2.1: Structure and example of a DID from [16]

A DID points to a DID-Document that stores the metadata, which has to be available to verify an identity. The DID-Document has the JSON-LD format and must contain a "@context" field which points to the version of DID specification used and an id field that shows the connected DID. DID-Documents might also include the fields "publicKey", "authentication", and "service". The "publicKey" section contains among others a DID-Reference, a type, and the actual public keys. This key might be used for cryptographic operations like verification of digital signatures and proofs or the

²w3c-ccg.github.io/.

2. Preliminaries

creation of secure channels. An authentication field specifies which public key can be used to prove that a DID is associated with a DID-Document. The service field might reference a service endpoint in a DID.

```
{
  "@context": "https://w3id.org/did/v1",
  "id": "did:example:123456789abcdefghi",
  "publicKey": [{
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "RsaVerificationKey2018",
    "owner": "did:example:123456789abcdefghi",
    "publicKeyPem": "-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n"
  }],
  "authentication": [{
    // this key can be used to authenticate as DID ...9938
    "type": "RsaSignatureAuthentication2018",
    "publicKey": "did:example:123456789abcdefghi#keys-1"
  }],
  "service": [{
    "type": "ExampleService",
    "serviceEndpoint": "https://example.com/endpoint/8377464"
  }]
}
```

Listing 2.2: Example of a DID-Document from [16]

In contrast to centralized identity creation, DIDs are in general created locally by the owner in accordance with a DID-Specification using a DID-Method. The DID-Documents are self-signed. To prove the ownership to a third party, it is not sufficient to only check the DID-Document's signature, but also to guarantee that the DID-Document can be retrieved by the associated DID.

DIDs are designed to hide its owner's real world identity. Nevertheless, a person has to be cautious to preserve privacy. For example, if a person is sharing information with other parties, it is possible to be tracked. Therefore, different identities can be created for different purposes to improve privacy and to prevent correlating information. Another way to find correlations is to crawl DID-Documents since they are publicly available on the distributed ledger (DL). To mitigate this type of privacy invasion, public keys and service endpoints should be changed in different DID-Documents and under no circumstances should confidential information be stored in DID-Documents. On the other hand, there are many use-cases where it is desired to connect one's real-world identity with a virtual one. DIDs do not support this, but they can be used along with verifiable claims described in section 2.3 to establish a connection between a virtual and a real-world identity.

2.3. Verifiable Claims

As discussed above in section 2.2, DIDs do not provide a native way to store real world identity information. Nevertheless, there are several use cases where it is essential to prove the real name, age or address with a digital identity. Verifiable claims are one solution. The idea is to have an authority issuing a claim equal to the process of signing an official document like a degree, contract or passport in the real world. Identically to the real world the signed document can be used to proof the correctness of the data. The concept of verifiable claims has been published by W3C verifiable claim working group³ in a draft [18] that is still “Work-In-Progress”.

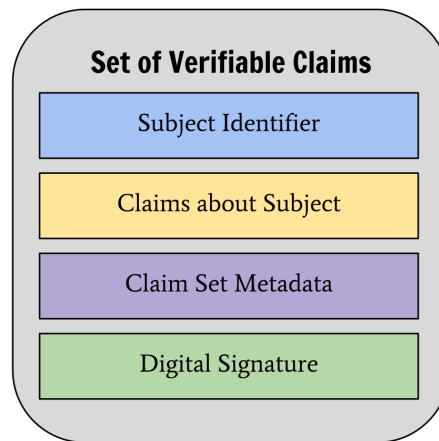


Figure 2.3.: Set of verifiable claims from [18]

2.3.1. Structure of Verifiable Claims

A subject is an entity like a person or organization about which a claim is made. A claim is a statement made by another entity that should be cryptographically verifiable. If an entity wants to make more statements about a subject, it would create several claims and combine them into a claim set. Figure 2.3 shows that a set of verifiable claims consists of a subject’s identifier, the claims, the claims’ metadata like the expiration date and a digital signature.

³www.w3.org/2017/vc/WG/.

2. Preliminaries

2.3.2. Verifiable-Claim-Ecosystem

The W₃C verifiable claim working group describes the four roles issuer, holder, inspector-verifier and identifier registry [18] and their relationship [19] in the so called “Verifiable-Claim-Ecosystem” shown in figure 2.4. The issuer is an entity that is responsible for issuing and revoking claims. This could be governments, corporations or universities in a real-world scenario. An issuer is communicating with a holder. An entity with this role could be a citizen, employee or student in a real world application, is controlled by a subject and it is acquiring and storing claims from the issuer. Furthermore, the holder might present a claim to an inspector-verifier. Examples for this role are security personal, insurances or websites that need verified information about a holder’s identity. The last role is called identifier registry and it is responsible for creating an identity on a holder’s behalf, or verifying identity information for an issuer or inspector-verifier. Examples for identity registries are corporate or government eID databases.

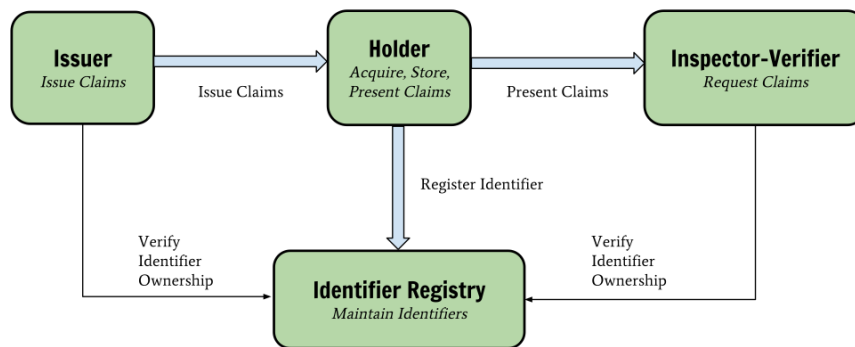


Figure 2.4.: Relationship between roles in Verifiable-Claim-Ecosystem from [18]

The workflow and the usage of verifiable claims are described in [19]. In the beginning, a holder must register itself to the identity-registry. Then the holder requests a claim from an issuer which verifies the subject’s identity information using the identity-registry and creates a claim. The holder receives and stores the verifiable claim. Next, the holder can use this claim to assert the identity information to some inspector-verifier. The inspector-verifier verifies the information and may use it for a corresponding application. In case of an error the issuer must be able to revoke the claim. The verification of a revoked claim would fail and therefore, a inspector-verifier would not accept it. Issuers, holders and inspector-verifiers can communicate using arbitrary channels or agents. As long as holders and inspector-verifiers trust the issuers, they can trust the

2. Preliminaries

verifiable claims as well. Due to, the architecture of the Verifiable-Claim-Ecosystem and the fact that holders are in control over their verifiable claims, there is no direct interaction between issuer and inspector-verifier which strengthens privacy, because an issuer does not know where the claim is used.

2.3.3. Data Model

Verifiable claims are based on the so-called entity credential model. This model consists of several properties in the form of name-value pairs. Mandatory properties are the id, an URI that represents the entity credential, a type, another URI that represents the class of the data set, and a claim that consists of an id referring the subject of the claim and at least one property. Optional parameters are the issuer's URI and the issuing date. If the claim should be verifiable, a signature property is needed. The same holds for revocation, if a revocation should be supported a revocation property is needed as well. An example of a verifiable claim can be found in the Appendix in Listing A.1. To enable a smooth interaction between all parties the W₃C verifiable claim working group requires the verifiable claims to be machine readable, but does not enforce a specific format.

2.4. Consensus Protocol

Many systems have to deal with decision finding processes. Traditionally, a central authority is responsible for making decisions. With the rise of distributed and decentralized systems, the different parties need to agree on common consent. For this purpose, a consensus protocol can be used. It allows the involved parties to vote for a decision. Nevertheless, with systems getting more complex, the chance of system failures due to incorrect processing, bad transmission, intruders, or computers crashes, is increasing. These failures can result in a wrong decision. Therefore, many consensus protocols have a fault tolerance, which means that up to a certain threshold faulty votes can be submitted, but the correct decision is still made.

2. Preliminaries

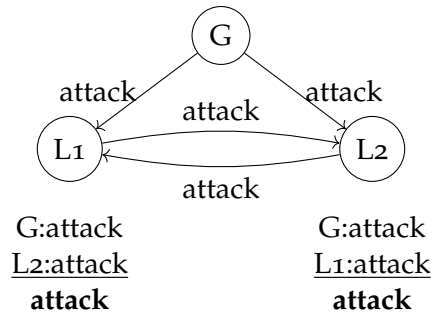


Figure 2.5.: Generals' Problem with three loyal nodes

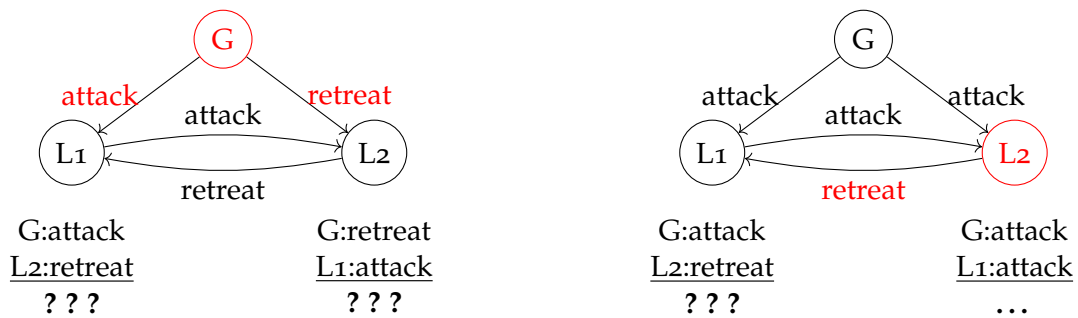


Figure 2.6.: Generals' Problem with two loyal lieutenants and a traitorous general

Figure 2.7.: Generals' Problem with one traitorous lieutenant

2.4.1. Byzantine Fault Tolerance

Already in the early 80s Lamport, Shostak, and Pease [20] investigated the fault tolerance and came up with the fictional Byzantine Generals' Problem: The generals are laying siege to a town. Their divisions are split up, and they are communicating using transmitters. They have to decide if they should attack or retreat. An attack can only be successful if it is coordinated, but some of the generals are traitors. The goal is that the loyal generals can launch a successful attack no matter if the traitors manipulate the decision finding process. Therefore, an algorithm is considered Byzantine Fault tolerant if it can produce correct results even though some parties are producing erroneous or malicious results.

The Oral Message (OM) algorithm [20] describes a solution for the Byzantine Generals' Problem. It looks at one General g which sends either an "attack" or "retreat" message to $n - 1$ lieutenants l where $n = g + l$. Each loyal lieutenant sends the received message

2. Preliminaries

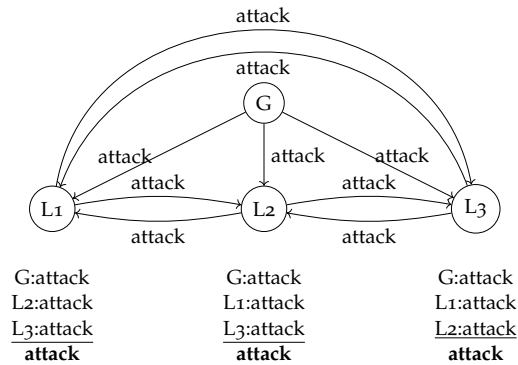


Figure 2.8.: Generals' Problem with four loyal nodes

to all other lieutenants. Traitorous lieutenants can have an arbitrary behavior and send any message. After receiving the messages from the other lieutenants, each loyal one creates a majority voting based on the messages it received.

For the OM algorithm, Lamport, Shostak, and Pease [20] assume that a recipient knows who the sender was and all messages are transmitted correctly. They give an upper boundary $3m + 1 \leq n$ for the algorithm, which states that more than two-thirds of the n nodes have to be loyal, where m is the number of traitors. This statement can be proofed by induction. If the number of traitors is lower than this boundary, all loyal lieutenants will obey the same order, and if a commander is loyal too, they will also obey the commander's order. Figure 2.5 shows a general with two lieutenants, all of them being loyal. Figure 2.6 and 2.7 show the scenario of one traitorous general and one traitorous lieutenant. In this cases, we have $3 + 1 \leq 3$, and we can see that the loyal lieutenants are unable to conclude.

Next, one node is added. The case of all parties being loyal shown in Figure 2.8 is still correct. The examples from Figure 2.9 and 2.10 show that $3 + 1 \leq 4$ is satisfied and the loyal lieutenants are coming to a conclusion.

2.4.2. Practical Byzantine Fault Tolerance Protocol

Castro and Liskov [21] introduced the Practical Byzantine Fault Tolerance Protocol (PBFT) which executes an operation on state-machine replications and tolerates Byzantine Faults. The number of acceptable faulty nodes f still has to be smaller than $3f + 1 \leq n$, where n is the total number of nodes. PBFT works in asynchronous envi-

2. Preliminaries

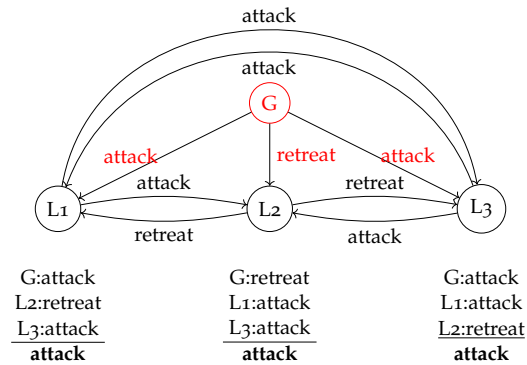


Figure 2.9.: Generals' Problem with four nodes and one traitorous general

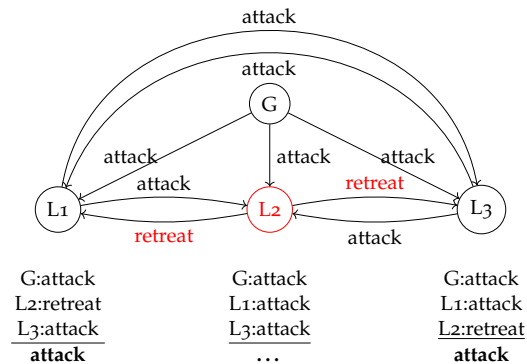


Figure 2.10.: Generals' Problem with four nodes and one traitorous lieutenant

ronments like the Internet and is designed for practical usage and real-world scenarios. The protocol starts with a signed request including a time stamp from the client that is sent to a node which owns the role of a primary. The primary creates a unique sequence number for the request and repackages it in the so-called pre-prepare message. Then the primary forwards the message to the backup nodes. Those nodes store the pre-prepare messages in their local log. Then the backups add their identifier to the content of the pre-prepare message to create prepare messages. Subsequently the backups multicast the prepare message to the replicas. If a replica receives a prepare message, it is saved and afterward compared with the content of the pre-prepare message. After $2f$ received prepare messages the replica changes its state to prepared and multicasts commit messages to the other replicas. If a node is prepared and has got $2f + 1$ commit messages, it alters the state to committed. Then it locally executes the operation and sends the result to the client. The protocol ends after the client received $f + 1$ responses.

2. Preliminaries

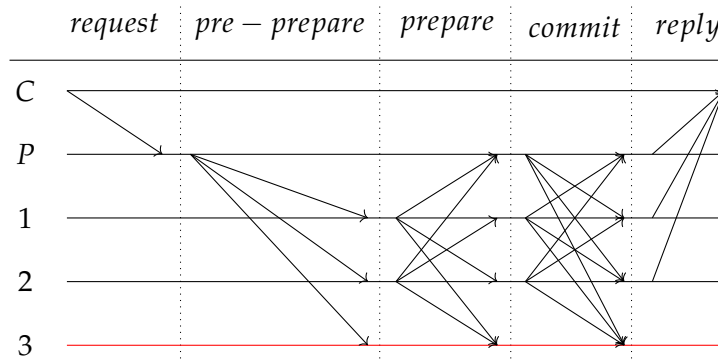


Figure 2.11.: PBFT execution with one faulty backup from [21]

The control flow of a PBFT process with a client, a primary, two working and one faulty backup is visualized in Figure 2.11. Due to PBFT's state machine replication technique, the protocol requires the client's operation to be deterministic and to have the same initial state for all replicas.

2.4.3. Redundant Byzantine Fault Tolerance Protocol

PBTF does not take a malicious primary into account, which holds back the message from the client for other nodes. To prevent a slowdown of the system or requests not being processed due to a malicious primary, Aublin, Mokhtar, and Quema [22] introduced the Redundant Byzantine Fault Tolerance Protocol (RBFT). RBFT runs $f + 1$ instances of PBTF on different machines with different primaries. One of them is the master instance, and the others are called backups. The backups monitor the throughput of the master while running in parallel. If the performance drops under a threshold, the master instance is assumed to be malicious, and a backup becomes the new master instance. RBFT implements an asynchronous open loop architecture which states that all requests are processed fairly and that clients are allowed to send multiple requests without waiting for the results.

The message flow is shown in Figure 2.12. In comparison to PBTF, the client sends the request to all nodes. They check the MAC and the client's signature and then propagate the message to all the other nodes. After receiving $f + 1$ messages the nodes start the PBTF processes with the same pre-prepare, prepare and commit steps. After a replica gets sufficient commit messages, the state is changed to ordered, and the node executes the operation and sends the result to the client. The client waits for $f + 1$ results.

2. Preliminaries

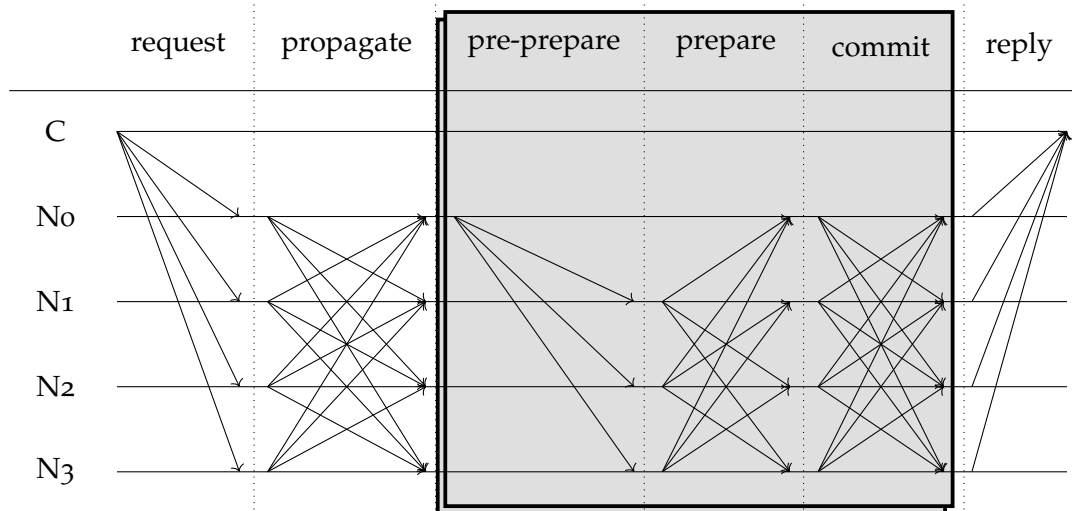


Figure 2.12.: RBFT execution with parallel instances shown in gray from [22]

2.4.4. Indy Plenum

Indy Plenum is an implementation of a Byzantine Fault Tolerance protocol. The code is open source and can be found on Github⁴. Indy-Plenum has been proposed by Law and Harchadani [23] and is based on RBTF. The two protocols differ in the cryptographic algorithms they use for inter-node communication: Plenum implements elliptic curve signatures instead of MAC-Authentication to supply non-reputation. Furthermore, RBTF does not specify any primary election process and any blacklisting algorithm. Plenum on the other hand has a deterministic and a non-deterministic voting mechanism and several blacklisting strategies that takes the severity of the fault into account. Finally, Plenum only requires the client to send the request to $f + 1$ nodes instead of a multicast to all nodes.

2.5. Self Sovereign Identities

The model of Self-Sovereign Identities (SSI) is the next step in the evolution of identity management systems [2]. The goal is to put the user in control of the identity data. The identity owner is responsible for deciding where the information is stored and with whom the data are shared. This approach contrasts the general practice of storing user-data in centralized registers. According to the Sovrin Foundation [24] the key

⁴github.com/hyperledger/indy-plenum/.

2. Preliminaries

requirements of SSI systems are besides privacy also performance, accessibility, and governance. The user has to trust the identity system. In contrary to centralized systems, the trust in SSI systems is decentralized across the network. Different parties are involved in the process of issuing and managing identities. On the one hand, there are endpoints for users that store their claims and keys locally in wallets and on the other hand, there are endpoints that are connected to the distributed ledgers (DL). A consensus protocol guides the decision finding process. There are two approaches for DL designs: either permissioned or permissionless. Permissionless states that everybody taking part in the DL environment can read and write on the ledger. Usually, a computational power intensive algorithm known as Proof-of-Work (PoW) ensures that a malicious party cannot change the ledger on its own. Permissioned means that only selected entities can read and write on the ledger, which needs less computational power. No matter which approach is implemented, the end-user either has to trust the PoW algorithm or the honesty of the nodes allowed to write on the ledger. Beside the communication between user and DL, users must be able to exchange messages with each other without a central entity. No matter which parties are communicating, the SSI system has to ensure secure message transmission and ways to verify the identity of the counterparty. Such identities must be unique across the system, but they must be created and stored without a central authority to prevent tracking. DIDs, described in section 2.2 can be used for this purpose. Furthermore, an SSI system needs a data format that holds statements about an identity. They have to be transferable across the network, and their authenticity must be trustworthy. Verifiable claims described in section 2.3 are intended for this goal.

An example of an SSI system is Sovrin [2] [24] [25]. It is under the umbrella of the Sovrin Foundation⁵, a non-profit organization that governs the network and the code development.

2.6. Hyperledger Indy

Hyperledger [26] is an open-source organization under the control of the Linux Foundation⁶. Its purpose is to be an umbrella organization for business-blockchain-projects. One of their projects is Hyperledger Indy, an implementation of an SSI system. In 2017 the codebase from Sovrin has been handed over to the Hyperledger Foundation [26]. The current approach is that the codebase is collaboratively extended under the control of the Hyperledger Indy project, whereas the Sovrin Foundation is governing the deployed network. Nevertheless, it is possible for private use, research and development to run Hyperledger Indy locally without access to the Sovrin network.

⁵sovrin.org/.

⁶www.linuxfoundation.org/.

2. Preliminaries

The implementation of the Hyperledger Indy project is abbreviated to Indy in this thesis. Indy contains code for client and server software. It defines how different parties are communicating and where the software is deployed and executed. The code further includes an implementation of DL technology, a consensus protocol, DIDs and verifiable claims.

2.6.1. Layers of Indy

An identity network is a complex structure with several abstraction layers. The concepts considered in Indy range from mathematical proofs in cryptography to the discussion of the defuse concept of trust. Therefore, the borders between abstraction layers are varying. Based on the explanation of the Sovrin trust network [25] [27] we explain the functionality of the framework and the components we are referring to in this work. A visualization of the interaction between different parties is shown in Figure 2.13.

Trust Layer

The Sovrin's trust layer is marked in blue in Figure 2.13 and specifies three different roles: trustee, steward and trust anchor. Since Sovrin is based on a permissioned ledger, meaning that only a selected number of nodes is allowed to operate on the ledger, guidelines and regulations that ensure trust have to be in place.

The trustee is the most highly privileged role in Sovrin, since identity owners assigned with this role make the decisions about the future of the Sovrin network. They are governing the ledgers and are the only ones allowed to post pool upgrade transactions. Trustees must be members of the board of trustees in the real world. The members of this board have to follow a specific selection process. Trustees are assigning new trustees, stewards and trust anchors.

Stewards are supposed to operate one or more Indy nodes. They have to fulfill legal requirements and must sign a legally binding agreement before being able to request a steward status. Stewards are supposed to be companies, universities, governments or other organizations that enjoy public confidence. Finally, stewards are involved in the voting process for adding new trust anchors to the system.

Trust anchors are entities for which trust is assumed. This includes stewards and trustees, but also identity owners who earned trust in the Sovrin network through the "Respect-Trust-Framework" [25]. Trust anchors are often organizations in the real world. They have two purposes: First, they are responsible for adding new Sovrin identities to the system and second, they are able to make a trust anchor recommendation where

2. Preliminaries

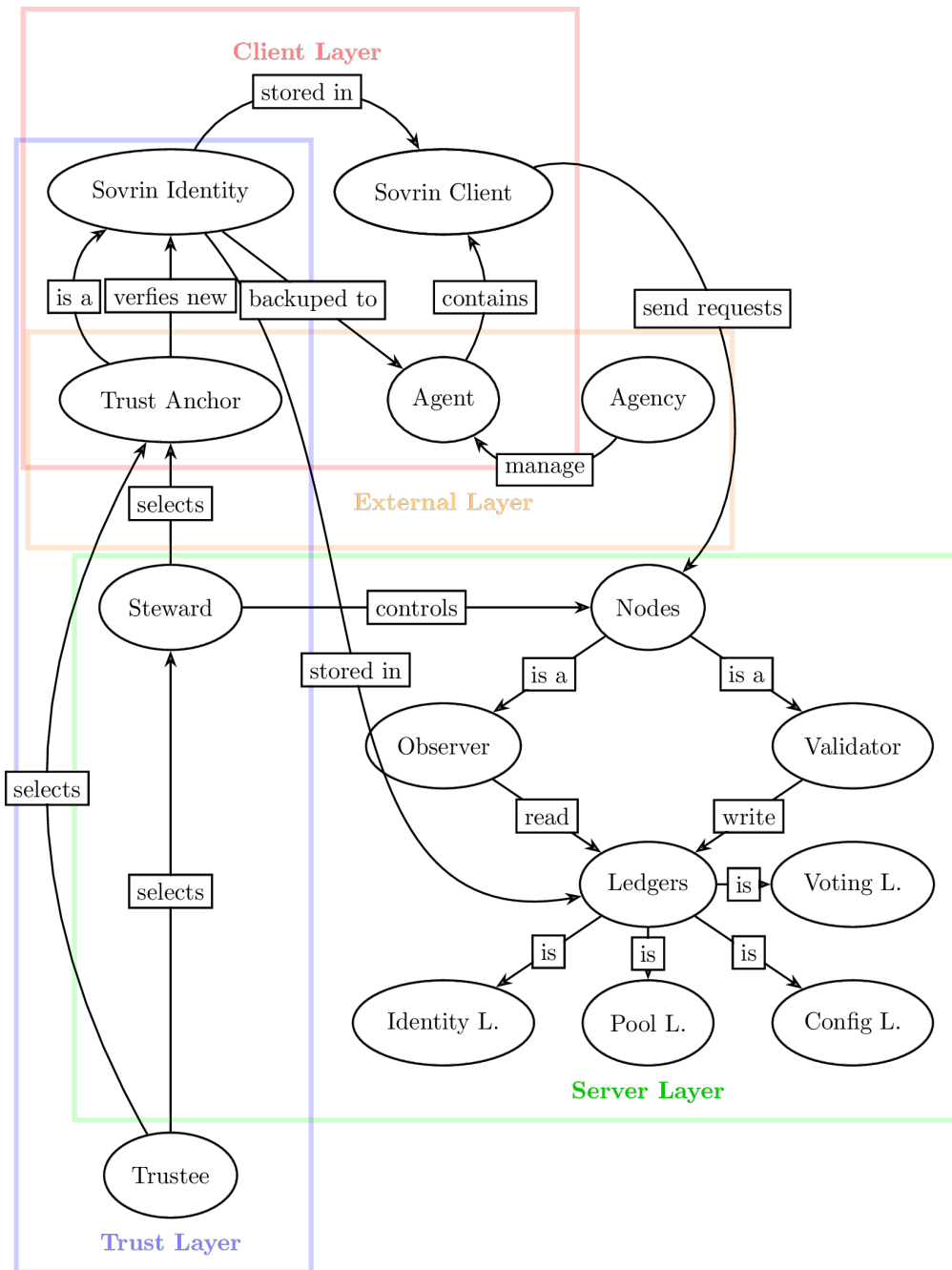


Figure 2.13.: Visualization of the most relevant parties in Indy

2. Preliminaries

they propose a Sovrin identity to the stewards for becoming a new trust anchor. Last but not least, the majority of Sovrin users are identity owners without a role, which represents an end-user who governs its credentials and Sovrin identities.

Server Layer

Sovrin is a decentralized identity network. Private data are stored locally, whereas public information is stored on multiple nodes across the system on DLs. The components related to the distributed ledger layer are kept in green in Figure 2.13. Only a selected number of nodes are permitted to make read or write transactions on the permissioned ledgers. Even though, only stewards are allowed to operate nodes, end-users may send requests to the nodes who query the ledgers for them. For this purpose, there are two types of nodes on the Sovrin network: validator and observer. Validators can handle read and write requests for the ledger. Further, validators have to ensure that they choose the correct ledger to write on and keep the ledgers up to date. Observers only process the users' read requests. Since those make up the majority of requests in the system, they can offload the validators and increase the overall performance of the system. Further, observers act as hot standbys in case of failing validator nodes, because they can easily be swapped into service instead of them.

Currently, the Sovrin identity network consists of four types of ledgers: the identity ledger, the pool ledger, the voting ledger and the config ledger. The identity ledger is the primary ledger, the location where the identity records of the Sovrin identity owners are stored. On the config ledger, the configuration data of the Sovrin network is stored. Data are set by the Sovrin Foundation technical governance board and approved by the board of trustees [25]. The voting ledger contains the votes of stewards and trustees. Based on these votes the pool ledger maintains a record of the voting outcomes and the resulting permissions for all nodes in the network. Sovrin clients can use the pool ledger to learn which nodes are observers and which are validators. Like in Ethereum⁷, the DLs in Sovrin are using states, which are stored in Merkle Trees [28]. The state-tree has to be the same across all nodes.

⁷www.ethereum.org/.

2. Preliminaries

Client Layer

The client layer is red marked in Figure 2.13 and shows the local setup for an end-user. An identity owner uses a Sovrin client to organize its Sovrin identities which utilize DIDs and verifiable claims (see section 2.2 and 2.3). Usually, the software is running on a private device, which ensures that confidential data are stored locally and that empowers the identity owner to stay in control of the personal credentials. A Sovrin identity has to be verified by a trust anchor before being able to use it on the Sovrin network. There is always a Sovrin identity underlying to a trust anchor, but not vice versa.

External Layer

The external layer is neither operated by a client nor by Sovrin, and it is kept in yellow in Figure 2.13. Sovrin identities can be backed up to a Sovrin agent. Agents provide an addressable endpoint for persistent Peer-to-Peer messaging and can be used for sharing identity data across devices like laptops and smartphones. Agencies create and maintain agents like web hosts provide email addresses or web spaces. Similar to web hosts, users can host agencies themselves, or use one from an already existing service provider. Even though agents provide additional functionality, it is possible to access the Sovrin network only using the Sovrin client. It is expected that there will be agencies in the future which already have a trust anchor status and which can automatically verify new Sovrin identities. The Sovrin foundation leaves the operation of agencies and further development of external services to the open market.

2.6.2. Technical Functionality

The Hyperledger Indy framework provides several functionalities on the client-side as well as on the node-side. The following section describes the most important use cases that the framework provides for our work. It is based on the Hyperledger Indy documentation [29] [30] [31].

2. Preliminaries

Client

Users run client software that is compatible with Sovrin. The software provides the possibility to store sensitive information like keys, DIDs, and claims in a SQLite⁸ wallet. The client can be used to connect the identity owner to other clients and the nodes. New DIDs can be created locally and published by sending a request to the nodes. Furthermore, clients can be used to create schema and claim definitions and to handle revocation and key recovery.

Nodes

The entries of DLs are called transactions. The nodes are responsible for writing and reading transactions to and from the ledgers. Usually, these processes are triggered by clients' requests, which are transmitted over the ZeroMQ⁹ transport protocol to arbitrary nodes. These endpoints check the transaction type in the request, to see, if it is a read or write request and choose an appropriate request handler. The request handler is implemented in all nodes and specifies how a request should be treated. These checks are semantical and syntactical. In case of a read request, the node queries its ledger and returns the result. For a write request, Plenum (see Section 2.4.4) is triggered. During the execution of the BFT Protocol, the request is multicasted over ZeroMQ to other nodes, and then the authenticity of the sender and the message are checked. If the protocol finishes successfully, the transaction is written to the ledgers and the states are updated. Otherwise, the request is rejected, and the ledgers remain untouched. In both cases the client receives a reply, indicating that the request succeeded or failed. The reply is only an acknowledgment. Therefore, a read request has to be sent after a successful write. It contains a verifiable state-proof, which ensures that the data has been written to the ledger.

A new transaction is stored in a log, and its hash is added to a Merkle Tree, which produces an inclusion proof for each new leaf. With every new transaction, the root hash changes as well. Alongside the ledger, the state must be updated as well. A state consists of a key-value pair and is stored in a Merkle Patricia Trie [32] [33]. This data structure can be used to prove that a key corresponds to a specific value when the root is known.

⁸www.sqlite.org/index.html.

⁹zeromq.org/.

2. Preliminaries

Ledger

The ledger is responsible for saving information that has to be publicly available. This includes DIDs, schemas, claim definitions and other public claims. Sovrin uses its own DID method which is called “sov” and results in identifiers like did:sov:21tDAKCERh95uGgKbJNHyp. The corresponding DID-Document is stored on the ledger and contains among others the Sovrin identity’s public keys and the agent’s endpoint for establishing a connection. Claims are based on a schema that describes the syntax and the attributes. A concrete implementation of a schema by an issuer is called credential definition. Schema and credential definition along with public claims are stored on the ledger. Public claims are essential for organizations which want to provide publicly available and verifiable information like business addresses or certifications. Personal claims and intelligence of individuals should never reach the ledger and remain locally.

2.6.3. Indy Workflow

The following section describes the most critical use cases that the framework provides for our work. This includes the process of adding a new identity, starting a communication channel with another user, creating a new credential definition and requesting a verifiable claim.

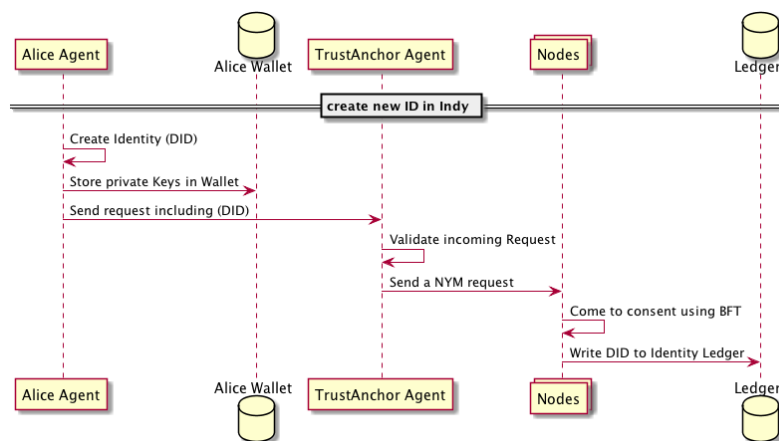


Figure 2.14.: A new identity is added to Indy

2. Preliminaries

Add new Identity

The process shown in Figure 2.14 is executed, to add a new identity in the form of a DID. An identity owner, in our case Alice, uses her agent to create a DID and store the confidential identity information in her wallet. She sends a request to a trust anchor, who validates Alice's identity and generates a "NYM" request that is sent to validators. These requests include the DID and a public key of Alice's newly created identity. The nodes run the Plenum BTF protocol and store the new identity on the ledger.

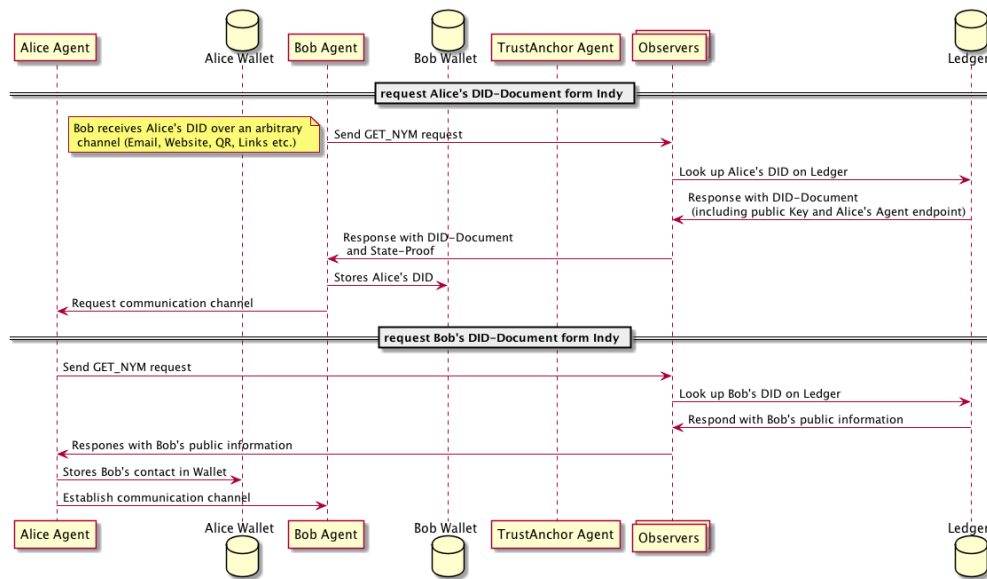


Figure 2.15.: Bob establishes a communication channel with Alice

Communicate with other Client

Figure 2.15 shows a scenario where two agents Alice and Bob try to establish a connection using the publicly available DID-Documents of each other. First, Bob receives a DID over an arbitrary channel. This can be a Link or QR-code in an email or on a website. Next, Bob sends a "GET_NYM" request including Alice's DID to an observer. The node queries the ledger and returns the corresponding DID-Document. Subsequently, Bob starts a connection with Alice's agent. Before communicating with Bob, Alice requests Bob's DID-Document from an observer using a GET_NYM request. Based on the received data, Alice verifies Bob's identity and use his public key for establishing a secure connection with him.

2. Preliminaries

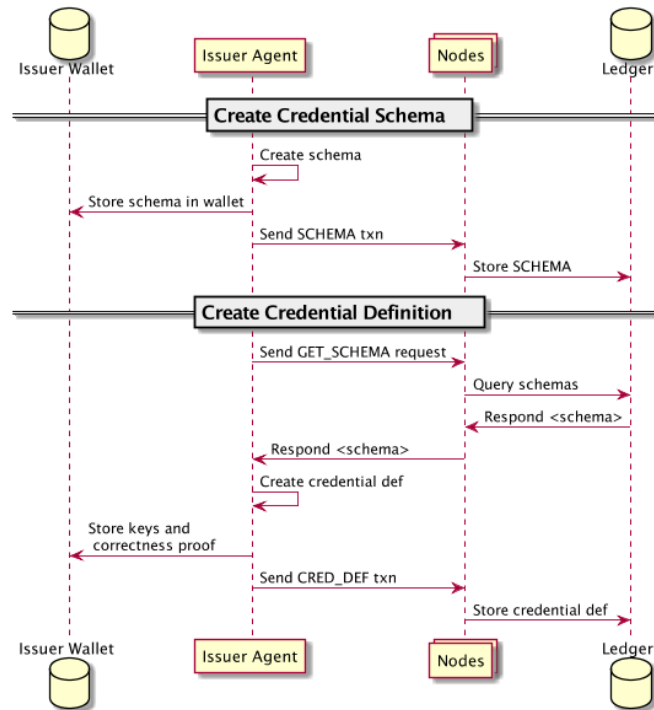


Figure 2.16.: An issuer creates a schema and a credential definitions based on [34]

Create Schema and Credential Definition

Figure 2.16 visualizes the process of creating a schema and a credential definition. First, an issuer uses the agent-software to create the schema. The schema is stored locally in the issuer's wallet. Furthermore, the issuer sends a "SCHEMA" transaction to the validators, who preserve the schema on their ledgers.

A credential definition is based on a schema and is essential if an agent wants to issue claims. If the issuer does not already have the corresponding schema in the wallet, a "GET_SCHEMA" request has to be sent to an observer. Once in possession of the schema, an issuer can create a credential definition, store the keys and correctness proof in the wallet and send a "CRED_DEF" transaction to the validators. These nodes keep a record of the definition in their ledgers after executing the BFT protocol.

2. Preliminaries

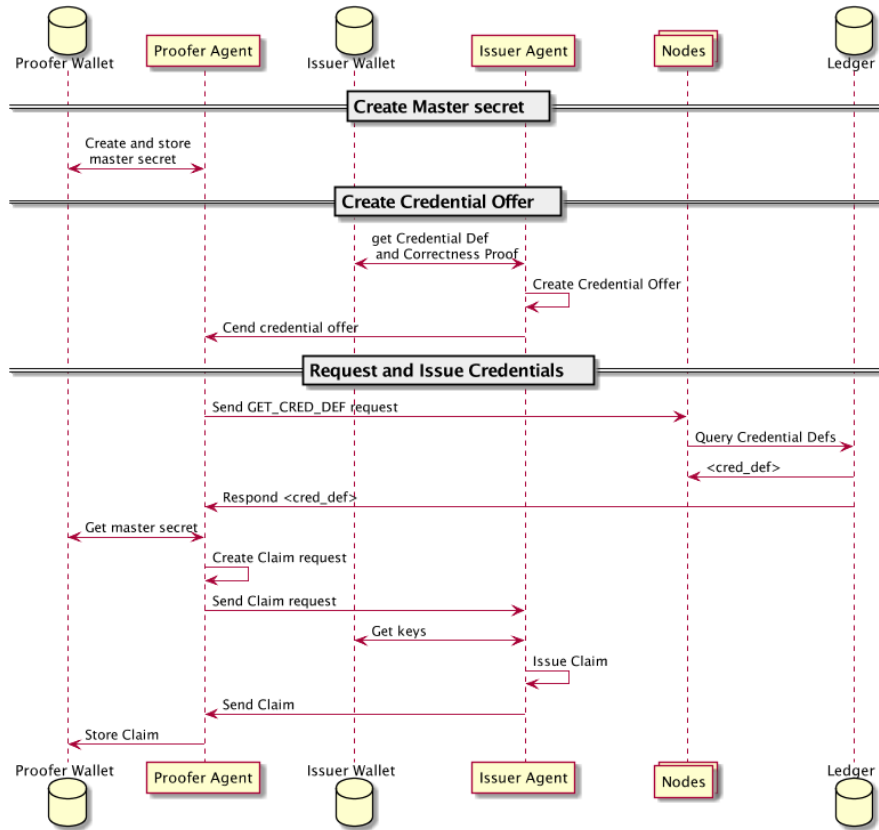


Figure 2.17.: A Proofer requests a claim from an issuer based on [34]

Request a Claim

Figure 2.17 shows the process of a proofer receiving a claim. First, a master secret is generated and stored in the proofer’s wallet. We assume that the proofer and issuer have already exchanged their identity data and can communicate with each other. Next, the issuer creates a credential offer based on the credential definition stored in the wallet and sends the offer to the proofer’s agent. Meanwhile, the proofer sends a “GET_CRED_DEF” request to an observer who returns the credential definition. The combination of definition and master secret allows the proofer to create a claim request, which is sent to the issuer, who uses the private key to generate a verifiable claim. Finally, the claim is returned to the proofer, who stores it in the wallet

2. Preliminaries

2.6.4. Codebase

The code of Indy is distributed under the open-source Apache license version 2.0¹⁰ on Github. The majority of code is written in Rust¹¹ and Python¹². The main repository is indy-node¹³, which contains the software for nodes and scripts to set up an identity network. Further, a lot of documentation is provided in this repository. Indy-node is dependent on indy-plenum¹⁴, indy-sdk¹⁵ and indy-crypto¹⁶. These projects are developed in separate Github repositories. Indy-plenum implements the Plenum BFT protocol and the DL. Indy-sdk contains an implementation of the anonymous credentials and verifiable claims, as well as software for the client. The SDK should help developers to build apps that use Indy. To support a wide variety of development environments, wrappers for .Net, iOS, Java and Python are provided. Indy-crypto is a shared cryptographic library that is based on the Apache Milagro cryptographic library¹⁷.

2.6.5. Multi-signatures

Indy supports the Boneh-Lynn-Shacham (BLS) [35] signatures-scheme. Due to the use of elliptic curves, the signatures are shorter than the one created with Digital Signature Algorithm (DSA) [36]. Therefore BLS is often regarded as “short signatures”. BLS signatures allow signature aggregations to multi-signatures [37]. Signing entities create signatures with distinct private keys and combine them into one signature. Multi-signatures demand all public keys of the signing entities for a verification. Therefore, a verifying party knows each entity who signed a message. In comparison to appending several individual signatures to a message, multi-signatures need less storage and guarantee that a message was signed as a group.

¹⁰raw.githubusercontent.com/hyperledger/indy-node/master/LICENSE.

¹¹www.rust-lang.org/.

¹²www.python.org.

¹³github.com/hyperledger/indy-node.

¹⁴github.com/hyperledger/indy-plenum.

¹⁵github.com/hyperledger/indy-sdk.

¹⁶github.com/hyperledger/indy-crypto.

¹⁷github.com/milagro-crypto/amcl.

3. Related Work

The following chapter describes technologies and implementations that are related to the ones we use. First, we show the current landscape of IdM systems that use blockchains and work out their similarities and differences. Next, we investigate related work that concentrates on Sovrin. Finally, we evaluate other projects that focus on identity derivation.

3.1. Identity Systems

Dunphy and Petitcolas [38] evaluated different identity systems with respect to the “Laws of Identity” [39]. Partially based on their findings, we investigate different identity systems that use blockchain technologies.

In this section, we evaluate ShoCard [40] [41] and uPort [42], two systems that utilize decentralized identity technology. A comparison of them with the Sovrin framework (see section 2.6) shows why we assume Sovrin to be the best fit for our requirements and why we decided to use it for our implementation (see chapter 5).

3.1.1. ShoCard

ShoCard [40] [41] is a closed-code identity solution that allows authentication to web-services with real-world-identities. Therefore, ShoCard imports an identity card via a mobile application into the user’s phone. The identity is verified, and the data’s hash is stored on the Bitcoin [43] blockchain. The mobile application can be used as a second-factor authentication or to reveal the real identity of a user to a web-service.

3. Related Work

The concept of Sovrin and ShoCard vary widely. Among others, the most notable differences are:

- Sovrin operates its own blockchain with the sole purpose of identity management, while ShoCard uses the Bitcoin ledger.
- In comparison to ShoCard, which is closed source and developed by the company ShoCard Inc., Sovrin is developed in the open-source project Hyperledger-Indy.
- ShoCard stores user information and uses its servers to connect users with services. They could collect knowledge about a user and analyze with whom a user communicates [38]. Sovrin, in contrast, relies on an architecture where a user is in control of the personal data and can communicate with other users and services in a decentralized fashion.

Due to the reasons mentioned above, ShoCard was not a suitable IdM system candidate for this thesis.

3.1.2. uPort

uPort [42] is an open-source project that is based on the Ethereum [44] blockchain and focuses on authentication. To start using uPort, a user creates a so-called uPortID using a mobile phone application. The id consists of a key-pair, where a user's phone stores the private key, while the Ethereum ledger preserves the public key. Besides the keys, a user can store identity data in a registry off-ledger, while the application links the data's hash to the uPortID on the ledger.

Sovrin and uPort follow similar concepts. DIDs and uPortIDs use asymmetric cryptography, sensitive data are stored locally off-ledger, and both support the use of claims. Nevertheless, their architectures differ in one aspect: the ledgers. Ethereum's distributed ledger is permissionless, while Sovrin's one is permissioned (see subsection 2.6.1). Due to selected nodes operating the ledger in Sovrin, the system has better scalability and higher throughput of transactions at no additional cost [38]. uPort, with the permissionless Ethereum blockchain, needs its own Gas crypto-tokens to run transactions, which means that all transactions come at an individual cost. Because of the advantages of the permissioned ledger and the trust network [27] that is built around it, we decided to use Sovrin instead of uPort for this thesis.

3. Related Work

3.2. Sovrin Integration

Sovrin's source code is distributed under the open-source Hyperledger Indy project. IBM and others [45] used indy's code and created the Verifiable Organizations Network¹ (VON), a digitalization approach of organizations from the Canadian Government in British Columbia. VON's goal is to issue and verify credentials for organizations and citizens, with the benefits of self-sovereign identities (see section 2.5). Even though the idea of deriving identity information from an official repository has similarities to the approaches presented in the concept of this thesis (see chapter 4), the initial situation is different and therefore, the VON network cannot be used for this thesis.

Sovrin is part of the Decentralized Identity Foundation (DIF)² along with other prominent members like Microsoft and IBM. The DIF tries to create industry standards for decentralized identity networks, by defining specifications and providing reference implementations. The DIF features many of the principles like DIDs and verifiable claims that Sovrin implements in its specifications. We conclude, that if we base our work on Sovrin, we implement state-of-the-art SSI approaches, which are becoming industry standards.

3.3. Identity Derivation

Identity derivation is a process that got more attention from governmental entities over the last years, especially with the rise of digitalization. The National Institute of Standards and Technology (NIST) in the United States presents a guideline for the derivation of personal identity verification credentials [46], which we cannot apply in this thesis, because of its strong focus on existing smart cards and our IdM architecture. In Europe two EU funded projects deal with deriving identities. The first one is the LIGHTest project³, which aims to build a global trust infrastructure across industries. They even specify their derivation scheme for mobile identities in [47], but the scheme cannot be applied for our derivation approach. The second European project is called ARIES project⁴. Its primary purpose is the creation of an ecosystem that derives identities to a secure wallet⁵. Even though this idea slightly correlates with the idea of

¹vonx.io/.

²identity.foundation/.

³www.lightest-community.org.

⁴www.aries-project.eu.

⁵www.aries-project.eu/content/aries-ecosystem.

3. Related Work

this thesis, ARIES does not use self-sovereign identities, and therefore, their schema cannot be adapted.

Due to, the lack of an existing approach for deriving identities from a system without a trusted third party, we came up with a new concept that we describe in chapter 4.

4. Concept

Our goal is to provide identity owners with qualified eIDs that they can use independently with all the advantages of an SSI system. Therefore, our concept provides an approach to import personal data from central authorities to local wallets of citizens, without losing the data's level of assurance. We aim to derive identities from trusted parties to an SSI network, where a user controls the identity data. In a decentralized environment, a place where everybody can claim to be anybody, we consider it as useful to have information that can be trusted and which can be used to ensure the relationship to a real-world identity. Our concept focuses on four steps. First, eIDs are retrieved from an IdP providing qualified data (see section 2.1). Second, trusted nodes run a consensus protocol to transform the data format. The process is decentralized, and the protocol ensures that malicious nodes up to a certain threshold cannot manipulate the data. Third, a user receives a signed claim (see section 2.3), which the recipient stores locally and can utilize for identity verification across the SSI system. Fourth, a revocation mechanism ensures that a claim is valid in the decentralized network as long as the original data from the IdP does not change.

We expect the traditional IdM to be able to issue qualified identity assertions and to provide a strong authentication mechanism to guarantee a high level of assurance. For the SSI system, we require support for verifiable claims. However, we do not assume that the data format of the two systems is the same. Therefore, this concept includes a transformation step, which converts the personal data into the new format without losing the level of assurance. In the following chapter, we describe in a step-by-step fashion our concept for the identity derivation and our approach to maintain trust by using the decentralized capabilities of an SSI as well as a revocation scheme.

4.1. Concept Architecture

For the derivation, we propose an architecture as shown in Figure 4.1. The existing system consists of an IdP that stores the user's eID and an identity interface which acts as a gateway for accurately transmitting personal data. The SSI system needs nodes that have the rights to write on the ledger. Further, we introduce a so-called identity

4. Concept

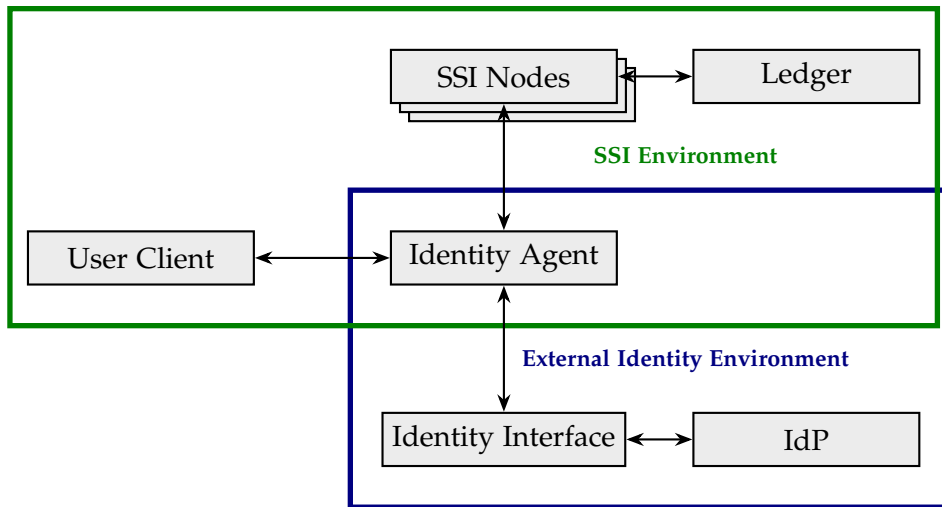


Figure 4.1.: Proposed architecture

agent, which implements an interface for both networks. Beside ensuring adequate message delivery between the traditional and the SSI system, the agent also acts as a gateway to the user.

4.2. Derivation

This section describes the identity derivation process that includes the data-format transformation. Initially, the user sends a request to the identity agent and is redirected from the identity interface to the IdP, which requires a login with strong authentication. After a successful login, the IdP sends the qualified identity assertion to the identity agent. This assertion includes several attributes describing the user's identity and a signature. Due to the usage of verifiable claims in JSON format in the SSI system, the data have to be transformed with high probability. The identity agent could perform the manual transformation, but the alteration is inevitable harming the integrity of the original signature. Without the signature, a verifier cannot trust the data anymore. Since the identity agent does not fulfill the same requirements as the original IdM system, the level of assurance decreases. Our concept provides a solution for this problem, shown in Figure 4.2. We take advantage of the SSI's decentralized trust network and the cryptographic properties of the BFT protocol (see section 2.4). We integrated the

4. Concept

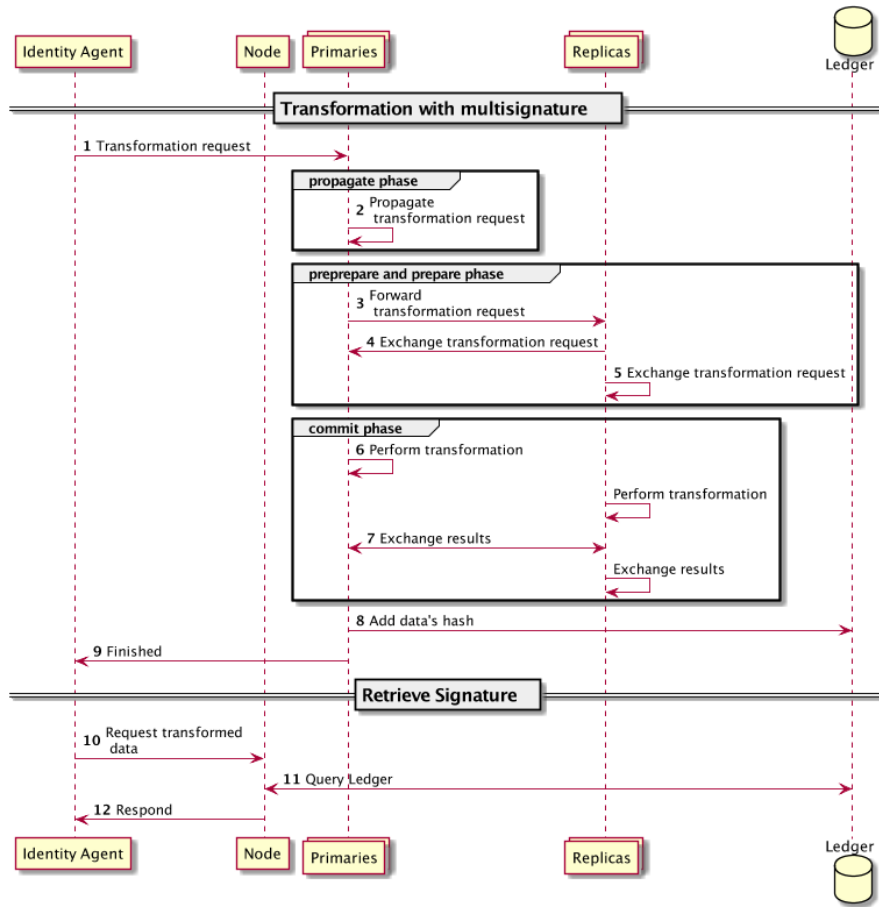


Figure 4.2.: Sequence diagram showing the identity transformation process

transformation process into the consensus protocol without changing the integrity preserving mechanisms of the algorithm. If a user trusts the SSI system, she shall equally trust the transformed data.

1. The identity agent checks the signature of the assertion. It extracts the personal information and creates a hash. Consequently, it generates a transformation request including the assertion, the personal attributes and the hash, which is subsequently sent to the SSI nodes who trigger the consensus protocol.
2. The request is further distributed across the primaries and replicas without checking the assertion or other content within the message.

4. Concept

3. Once the nodes received enough acknowledgments from each other, they enter the commit phase where the transformation is executed locally in each node.
 - (A) The node verifies the assertion's signature to ensure integrity and authenticity.
 - (B) Next, the node extracts and hashes the attributes from the assertion and checks if the generated hash matches the one in the request.
 - (C) In this case, the node creates a signature using the private key and subsequently sends a commit-acknowledge including the signature to the replicas and the primary.
4. The node waits for a more commit-acknowledges than the threshold of the consensus protocol demands (see section 2.4). If this is the case, the node aggregates the signatures to a multi-signature (see subsection 2.6.5) and stores the hash and the signature on the ledger.
5. To conclude the protocol, the identity agent receives a final acknowledge that indicates if the execution was successful or not.
6. Next, the identity agent sends a request containing the hash to an arbitrary node in the SSI network. The node queries the ledger and responds the multi-signature and the proof of a successful derivation.
7. Finally, the identity agent creates a verifiable claim (see section 2.6.3) including the personal attributes, the hash, and the multi-signature. The claim is transmitted to the user and stored locally in a secure wallet.

A user with a claim can share the verified personal information using the distributed architecture of the SSI system. Additionally to the normal process of verifying claims, a recipient must also check the multi-signature to ensure the eID's origin.

4.3. Revocation

In case the identity information of a claim has expired, a revocation mechanism is needed. Usually, identity assertions are short-lived and expire after a few minutes. Therefore, IdMs do not provide revocation mechanisms for them. Since the transformation and claim creation process creates a new signature that does not automatically expire, we need to find a method that checks if the data from the IdP have changed and then revoke the claim in the SSI system.

4. Concept

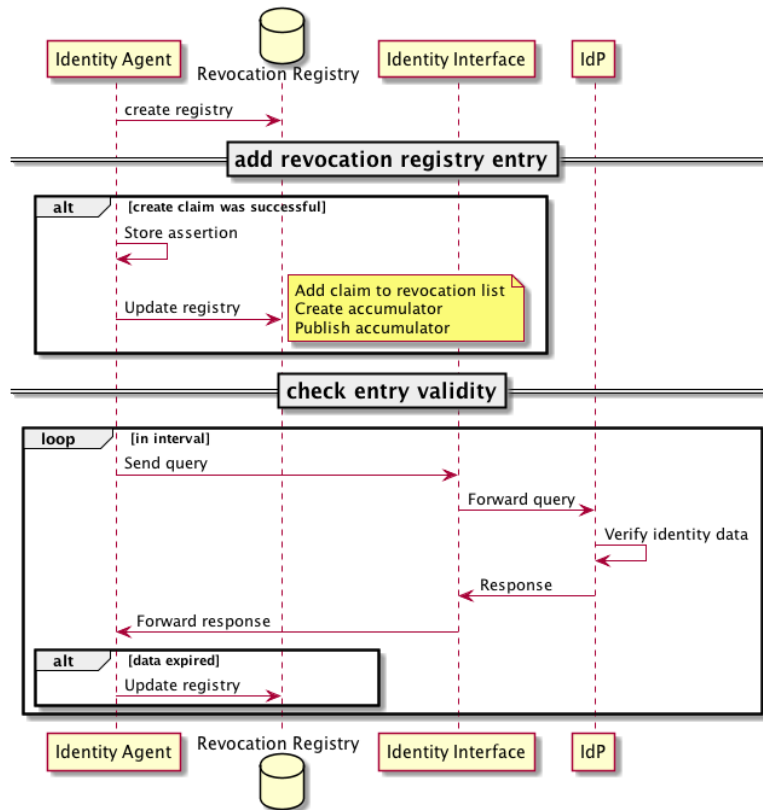


Figure 4.3.: Sequence diagram showing the concept for revocation

In our concept, the identity agent is responsible for managing the revocation list and to keep it up-to-date. Therefore, the agent has to frequently check if the data retrieved from the IdP are still valid. Therefore, we propose the method shown in Figure 4.3 to check the validity. First, the identity agent has to store the identity assertion. Second, the agent sends the assertion along with an attribute query over an identity interface to the IdP. Since no further data are requested, no privacy problem occurs. Third, the identity interface is informed by the IdP if the data are still valid, which it forwards to the identity agent. Forth, if the data are expired the identity agent changes the revocation list.

5. Implementation

The following chapter describes how we implemented our concept, explained in chapter 4, in an executable PoC. We explain the implemented interfaces and our setup. Our solution uses the eIDAS network as a source for qualified eIDs and Hyperledger Indy as an SSI system where we import the identity. Since we deal with existing systems, we take advantage of specific functionalities of these systems. Therefore, we give the components in the implementation the names they have in the original systems to prevent confusions with our generic concept. The relationship of the components in this implementation with the ones proposed in our concept is visualized in Figure 5.1. We focused on components connecting the two systems. Namely, the client agent, eIDAS agent and the extended fault tolerance protocol implemented in the validator nodes. The other components remain untouched in our implementation and, therefore, are not further explained in this chapter.

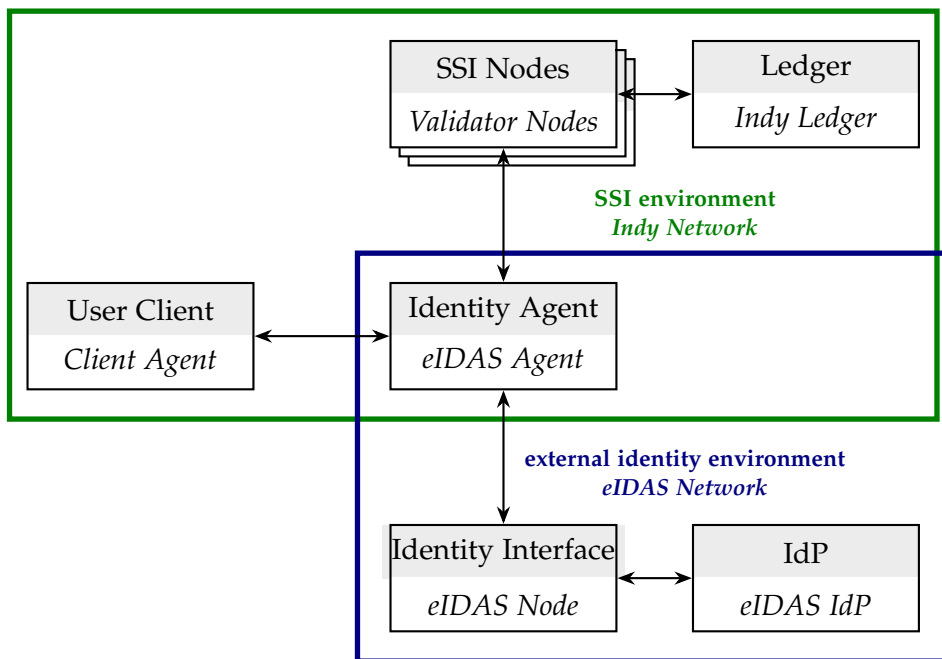


Figure 5.1.: Relationship of components in concept (see chapter 4) and PoC

5. Implementation

5.1. Client Agent

The client agent forms the entry point for a user to the Hyperledger Indy SSI and offers functionality for sending and receiving data [48]. Since the system is per definition decentralized, a user must have an addressable endpoint to be reachable for other parties. Our client agent uses DIDs (see Section 2.2) that specify an address and a public key for secure communication. Further, the agent must be able to handle verifiable claims (see Section 2.3), because the derived eID is stored as a claim. More specifically the client agent implements the following functionality:

1. **Request DIDs from other parties to establish a communication channel (see section 2.6.3):** A DID consists of an identifier, a public, and a private key. Usually, the identifier starts with an identifier idstring, which is per-default “did:plenum” in Indy. This string is crucial if different SSI platforms share DIDs. We do not use an identifier idstring in our implementation since our PoC works in a closed environment and we do not intend to distribute claims issued from our prototype. Listing 5.1 shows the current DIDs and the public key contained in the DID-Document of a trust anchor and a steward (see section 2.6.1). To retrieve these DID-Documents, a user must send a so-called “read NYM” request to an arbitrary node, a function the Java wrapper of indy-sdk provides.

```
Issuer (Trustanchor)
DID: AgQbxDbdWX9KFRscaYrT4C
Verkey: 6GzBT81CdQ88DJEjt4VKxBCrvR13XVPDDLKSVR3Nzt5b

Steward
DID: Th7MpTaRZVRYnPiabds81Y
Verkey: FYmoFw55GeQH7SRFa37dkx1d2dZ3zUF8ckg7wml7ofN4
```

Listing 5.1: DIDs and public keys of an issuer and a steward.

```
User (runs client agent)
DID: AtgRBBjATvuSV1U7p4iyeb
Verkey: 6Pg9JRnUou6iGv8QaNWNj4RJUCsYZqiDpP6UkeUEtT4Z
Signingkey: 27XD5Qg4ed4tb5kwqeZLd4FrN1rK1EX...N3HeCX9dh1X
```

Listing 5.2: DID and keys of a user

2. **Create and register a DID (see section 2.6.3):** The client agent locally creates the user’s DIDs. Listing 5.2 shows the identifier, the public key called “verkey” and the private key called “signingkey” of a user. Once created, the nodes should add the DID-Document to the ledger to make it publicly available. Listing 5.3 shows Indy’s standard write NYM request and response [48]. The messages contain the user’s DID in the “dest” field and the public key in the “verkey” field. The identifier field corresponds to the steward’s DID, who holds the privilege of

5. Implementation

operating nodes and writing new DIDs to the ledger. The response contains the same attributes, in addition to fields proofing that the DID has been written to the ledger. The “auditPath” characterizes a cryptographic proof that the identifier has been appended to the ledger. The “rootHash” relates to the ledger’s Merkle tree and the “txnTime” represents the timestamp when this action took place.

```
NYM Request
{ "reqId": 1541684803116063042,
  "identifier": "Th7MpTaRZVRYNpiabds81Y",
  "operation": {
    "dest": "AtgRBBjATvuSV1U7p4iyeb",
    "type": "1",
    "verkey": "6Pg9JRnUou6iGv8QaNWNj4RJUCsYZqiDpP6UkeUEtT4Z"
  },
  "protocolVersion": 1
}
NYM Response
{ "seqNo": 30,
  "signature": "2F95RJMGED1HTK6m2LmTWJjCYx8LPRkW...HBuX3EZ",
  "dest": "AtgRBBjATvuSV1U7p4iyeb",
  "verkey": "6Pg9JRnUou6iGv8QaNWNj4RJUCsYZqiDpP6UkeUEtT4Z",
  "reqId": 1541684803116063042,
  "auditPath": [
    "7847ThYvEyMk2wSVzTgTXBj1fvyeZqBzukupQEargrcGz",
    "AJmbFzQQGD8QdkAGDragFqvkl6QEbsdtiJYakjusGpES",
    "8CaZQ1S7NMHQsWyZkp1v2T74JuvvwXEx857YJ382wkJ2",
    "CRfxF2NNP7FqJGFJ1rj39RqoY4cckgYd6VjN1AHE6M4Q"
  ],
  "type": "1",
  "rootHash": "Tn63XLzqTbw7Xsg9syPmBs6waWupordpUZvuMhz8rfr",
  "txnTime": 1541684803,
  "identifier": "Th7MpTaRZVRYNpiabds81Y",
  "signatures": null
}
```

Listing 5.3: NYM request and response

3. **Request a claim from a trusted party (see section 2.6.3):** Indy does not specify how the claim creation is triggered. We decided to inform the eIDAS agent after a successful authentication. The eIDAS agent operates a trust anchor and sends a claim invitation to the client. A handshake between the two agents follows where they verify their identities and exchange information. Finally, the trust anchor sends the claim to the client agent.
4. **Store the private keys and claims that relate the user’s DID:** Our agent creates a new DID for each execution of the demonstrator. It further stores the corresponding private keys and the user’s claims in an SQLite database, which we delete immediately after showing them to the user. Since we deployed the agent for public use on a server, the database is located on the server. Even though we would

5. Implementation

not advise storing confidential data on the server-side in a non-conceptual implementation, we decided to use this approach for more simplicity and usability in the PoC.

5. **Trigger an identity derivation process:** A user triggers the derivation process with the click of a button in the Java web application on a JSP-page.

Functionalities 1 to 3 are part of the capabilities of Hyperledger Indy. We used the `indy-sdk` to compile a shared library called `libindy`. The framework written in Rust further provides APIs and wrapper functions for various programming languages to access the `libindy` conveniently. A user must operate a client software to access the Indy network. At the time of writing this thesis, the community is developing an application that a user executes locally and which provides all the functionality needed for long time use. For our PoC, we aimed for an easy to use demonstrator. Forcing a user to install and setup software locally before being able to use the demonstrator was not a convenient solution for us. Therefore, we decided to use the Java wrapper from the `indy-sdk` to build our client agent into a Java web application.

5.2. eIDAS Agent

The eIDAS agent forms the center of this PoC by connecting the eIDAS environment with the SSI-network and ensuring the transmission of data between the two systems. Besides this duties, the eIDAS agent is responsible for providing the user with an interface for triggering an import and issuing new claims. For connecting the eIDAS agent with the eIDAS network, we modified an eIDAS Service Provider (SP) stub from the eIDAS-Node bundle version 1.4¹ and connected our modified SP with a demo eIDAS node. The eIDAS SP stub is a web application written in Java. Therefore, we used the Java wrapper of the `indy-sdk` and deployed our final web application on a Tomcat 8. The `indy-sdk` provided us with tools to set up a basic indy-network inside of docker, as well as other functionalities to manage DIDs and claims.

Like the client agent, the eIDAS agent forms an addressed endpoint in the indy-network. Due to this, we used the `indy-sdk` to create a DID. In comparison to an agent operated by a private user, the eIDAS agent is part of a type of agents whose addresses and identities are publicly available. They are generally operated by public organizations

¹ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/eIDAS+Node+version+1.4.

5. Implementation

like universities, state departments or companies. In our PoC the eIDAS agent represents the entry point for the import from the eIDAS network. Therefore, its address is publicly known and the agent is available for everybody in the network. Since we created our own indy-network for this PoC, entities from other SSI networks cannot reach the eIDAS agent.

```
{
  "ver": "1.0",
  "id": "Th7MpTaRZVRYnPiabds81Y:2:eidasimport:1.0",
  "name": "eidasimport",
  "version": "1.0",
  "attrNames": [
    "familyName",
    "proof",
    "personIdentifiertype",
    "dataOfBirth",
    "givenName",
    "eidasloa",
    "hash"
  ],
  "seqNo": null
}
```

Listing 5.4: A schema created by a steward

A schema and a credential definition must be stored on the ledger before the eIDAS agent can issue claims. These JSON constructs refer to the issuing entity and specifies among others the attributes included in future claims. We created a schema that consists of the hash and a proof besides personal fields, like the name and the date of birth. To store schema and credential definition on the ledger, they are previously sent to indy-nodes using Indy's "SCHEMA" and "CLAIM_DEF" requests from the indy-sdk. Listing 5.4 shows an Indy "SCHEMA" request. Besides the personal attributes in "attrName", the "id" tag "Th7MpTaRZVRYnPiabds81Y:2:eidasimport:1.0" includes information about the schema's name "eidasimport", the version number "1.0" and the issuer's DID "Th7MpTaRZVRYnPiabds81Y", which relates to the steward. Since credential definitions are implementations of a given schema, there must be a reference. Therefore, the credential definition's "id" tag "AgQbxDbdWX9K FRscaYrT4C:3:CL:Th7MpTaRZVR YnPiabds81Y:2:eidasimport:1.0" in Listing 5.5 contains the schema's id as a subset. The other part contains the DID of the entity which is responsible for the credential definition. In our case, the trust anchor issued it. The definition further states "CL" as a "type", which specifies the Camenisch and Lysyanskaya signature scheme [49] for the claim creation process. The scheme uses the numbers in the "primary" for the encryption.

5. Implementation

```
{
  "ver": "1.0",
  "id": "AgQbxDbdWX9KFRscaYrT4C:3:CL:Th7MpTaRZVRyNPiabds81Y:2:eidasimport:1.0",
  "schemaId": "Th7MpTaRZVRyNPiabds81Y:2:eidasimport:1.0",
  "type": "CL",
  "tag": "tag1",
  "value": {
    "primary": {
      "n": "11950222728048176487899999417300410284360103281...3442642546549",
      "s": "27836975285873105452939568128446675516587331159...1441629756178",
      "rms": "333185308451795223834910739726375650060250010...0118262489522",
      "r": {
        "familyname": "160110742647689620584489969607881400...0665142352504",
        "givenname": "4720946505133189971230246420069883624...3464227659042",
        "proof": "11030982820946311923413789091588006783545...3874496544494",
        "hash": "331831109619902809583611304043511214544138...2271254377164",
        "dataofbirth": "19653671989362264456966326639902362...0885058066848",
        "personidentifiertype": "45898686385124485781949742...5995388914290",
        "eidasloa": "36214432088611070540344343039944099817...0202473665801"
      },
      "rctxt": "6124582150853602924026242362878310529211585...3173201252207",
      "z": "93318415286280718552370015642301205966218646035...8970208250366"
    }
  }
}
```

Listing 5.5: Credential definition created by an issuer

Since the eIDAS agent orchestrates the import process, the following enumeration will describe this process from the view of an eIDAS agent.

1. We created a JSP-page with a button to start the process.
2. Once a user clicks on this button, the agent creates a SAML-request that looks like Listing 5.6. We defined the return-URL and the SAML-Metadata of the eIDAS agent to ensure that the response is returned to our SP. The request further features a list of attributes we want to retrieve from the eIDAS system. Finally, the XML structure contains information about the encryption like the signature methods supported and the certificate used.
3. Following a successful creation, the eIDAS agent redirects the request and the user to the demo eIDAS node. There the user can choose between using an Austrian mobile phone signature or the credentials for a test-user for authentication.
4. Once the authentication is complete, the eIDAS node forwards the user accompanied by a SAML-assertion in Listing 5.7 back to the eIDAS agent. Like the request, the assertion contains information about the signature scheme and cryptographic parameters used, along with the user's personal information inside the "saml:AttributeStatement" tag. Our agent validates the request's signature and then starts the transformation process.

5. Implementation

```
<saml2p:AuthnRequest Destination="https://vidp.gv.at/eIDAS_node/eidas/
  ColleagueRequest" >
  <saml2:Issuer>http://importdemo.iaik.tugraz.at/SP/metadata</saml2:Issuer>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:CanonicalizationMethod "http://www.w3.org/2001/10/xml-exc-c14n#" />
    <ds:SignatureMethod "http://www.w3.org/2001/04/xmldsig-more#rsa-sha512" />
    <ds:SignatureValue>dofilLbSlc7mT2y...SbHFqEzqSgE=</ds:SignatureValue>
    ...
    <ds:X509Certificate>MIIFQTCCAy...LV1UFuCCe=</ds:X509Certificate>
  </ds:Signature>
  ...
  <eidas:RequestedAttributes> ...
  <eidas:RequestedAttribute Name="http://eidas.europa.eu/attributes/
    naturalperson/CurrentFamilyName" ... isRequired="true"/>
  <eidas:RequestedAttribute Name=".../CurrentGivenName" isRequired="true"/>
  <eidas:RequestedAttribute Name=".../DateOfBirth" isRequired="true"/>
  <eidas:RequestedAttribute Name=".../PersonIdentifier" isRequired="true"/>
  </eidas:RequestedAttributes>
  ...
  <saml2:AuthnContextClassRef>http://eidas.europa.eu/LoA/high</saml2:...>
  ...
</saml2p:AuthnRequest>
```

Listing 5.6: Shortend SAML-request of Listing A.3 in the Appendix

5. The eIDAS agent extracts the personal information from the SAML-assertion and puts it in a predefined order in a JSON document and subsequently calculates a hash using SHA256 [50].
6. We create a custom Indy request, which we call “samltransformer-request” (see Listing 5.8). This request in JSON format contains the hash and the original SAML-assertion in the payload.
7. Since our multi-signature creation is running in a python environment, we use shell scripting and python to start the validator nodes.
8. Once the nodes have created the multi-signature, they send an response to the eIDAS agent. If the message reports a successful transformation, the agent sends a “get-saml-transformer-request” (see Listing 5.9) including the hash to an arbitrary Indy node.
9. After the node crawled the ledger and returned a state proof including a multi-signature, the agent exits the python environment and continues in Java.

5. Implementation

```
<saml2:Assertion ... >
<saml2:Issuer>https://vidp.gv.at/eIDAS_node/eidas/metadata</saml2:Issuer>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod "http://www.w3.org/2001/10/xml-exc-c14n#" />
    <ds:SignatureMethod "http://www.w3.org/2007/05/xmldsig-more#sha256-rsa-
      MGF1" />
    <saml2:SubjectConfirmationData Address="129.27.142.188"
      Recipient="http://importdemo.iaik.tugraz.at/SP/ReturnPage" />
    ...
    <saml2:Conditions NotBefore="2018-11-08T09:25:06.654Z"
      NotOnOrAfter="2018-11-08T09:30:06.654Z">
      <saml2:Audience>http://importdemo.iaik.tugraz.at/SP/metadata
    </saml2:Audience>
    ...
  </saml2:Conditions> ...
  <saml2:AuthnContextClassRef>http://eidas.europa.eu/LoA/high</saml2:...>
  ...
<saml2:AttributeStatement>
  <saml2:Attribute Name="http://eidas.europa.eu/attributes/naturalperson/
    CurrentFamilyName" > ... Mustermann ... </saml2:Attribute>
  <saml2:Attribute Name=".../CurrentGivenName">...Max...</saml2:Attribute>
  <saml2:Attribute Name=".../DateOfBirth">...1940-01-01...</saml2:Attribute>
  <saml2:Attribute Name=".../PersonIdentifier">...AT/SO/MiACFWU9...
    CXDb1W1gv4=...
  </saml2:Attribute>
</saml2:AttributeStatement>
</saml2:Assertion>
```

Listing 5.7: Shortened version of the SAML-assertion in Listing A.4 of the Appendix

10. We subsequently use the Java-wrapper of indy-sdk and our formerly created schema to create a claim shown in Listing A.2 of the Appendix.
11. The eIDAS agent finally hands the claim over to the client agent and redirects the user to a webpage where the claim is displayed along with attributes from the assertion.

5.3. Validator Nodes

Validator nodes store copies of the ledgers and may write on them. Due to this privileges, the Sovrin Foundation has a strict nomination process for allowing the creation of new validator nodes, which should guarantee a maximum of trust (see section 2.6.1). The Indy SSI does not work without trusting these nodes. Therefore, we decided that the validator nodes should determine if the transformation from the XML structure of the SAML-assertion to the JSON format of the verifiable claims worked as intended, and the personal information was not altered. We use the consensus protocol plenum to neglect small numbers of malicious or faulty nodes. We based the validator nodes

5. Implementation

of our PoC on the code from the Github repository `indy-node` and our customized BFT protocol on `indy-plenum`. We wrote this part of the PoC in Python 3.6 because `indy-node` and `indy-plenum` use this language.

We used a plugin structure of `indy-plenum` to extend the protocol with the functionality to accept our custom requests and to validate the input accordingly. This so-called requesthandler defines the format of read and write requests and allows checks in each phase of the BFT protocol. Besides the request handler, we created a new ledger with the id “42”. The ledger only serves the purpose to record successful transformations and to store multi-signatures. To maintain the ledger’s state, we had to customize Indy’s state creation function and to create an encoding and decoding function, which is further used for the transmission of messages across nodes.

```
{
  TXN_TYPE: 42420,
  DATA: {
    'hash': 'eed456f7a10fd942c256644330c590e88d82ae87ff2b76b75afdb392d4bcaa8f',
    'saml': 'eed456f7a10fd942c256644330c590e88d82ae87ff2b76b75afdb392d4bcaa8f.xml'
  }
}
```

Listing 5.8: Write `saml-transformation-request`

The multi-signature creation starts with a write “`saml-transformation-request`” shown in Listing 5.8 that the eIDAS agent sends to an arbitrary validator node. The node evaluates the message’s identifier to find the corresponding requesthandler. If the write request has the identifier “42420”, the node selects our custom “`saml-requesthandler`”. Once the node selects this requesthandler, it subsequently checks if the message contains a SAML-assertion and a hash and starts, in this case, the BFT protocol. Our version of the protocol keeps the original phases of plenum, and therefore messages are distributed as usual using ZeroMQ to backups, primaries and replicas in the propagate, pre-prepare and prepare phase without our interfering. Once the commit phase is reached, each primary and each replica executes our code to check the integrity of the received data and performs the transformation on its own. The following checks are performed locally:

1. Every node utilizes certificate storages containing certificates of the eIDAS nodes. They use those certificates to verify the signature of the eIDAS SAML-assertion contained in the request to ensure that the message originated from eIDAS and had not been altered.
2. Next, every primary or replica extracts the personal information from the assertion.
3. They create a JSON and put the attributes in the same order as the eIDAS agents.

5. Implementation

4. Finally, each node hashes the personal information and checks if it is the same as the hash contained in the request. In this case, the node signs the JSON document using the Boneh-Lynn-Shacham (BLS) (see 2.6.5) scheme and sends a commit message including the signature to the other replicas.

```
{
  TXN_TYPE: 42421,
  DATA: {
    'hash': 'eed456f7a10fd942c256644330c590e88d82ae87ff2b76b75afdb392d4bcaa8f'
  }
}
```

Listing 5.9: Read get-saml-transformation-request

Once more than two-thirds of the nodes have come to the same result and have received other commit messages, they enter the reply phase. There, we create a multi-signature from the combination of single BLS signatures and store them on the ledger. Since the data on the ledger is publicly available, we only preserve the hash and no other data related to the personal information from the SAML-assertion. Finally, the nodes update their states, create a state-proof and return an acknowledge to the sender, which is the eIDAS agent in our case.

After the eIDAS agent has received an acknowledge, it sends a read “get-saml-request” as shown in Listing 5.9. The receiving node knows due to the message’s transaction-type-number that it should use our “saml-requesthandler”. There, we specified that in the contrary to a write request the BFT protocol should not be started. Instead, recognizing the identifier “42421” as read request, the node will extract the hash from the request and query its ledger to find the associated multi-signature. Once found, the node returns the multi-signature and a state-proof that guarantees that the hashed data was part of the ledger. The eIDAS agent will use this data to create a verifiable claim, but the same procedure can be used by anybody in the network to ensure that the trusted validator nodes have processed the personal data and can, therefore, ensure the trustworthiness of the information.

6. Demonstrator

The following chapter shows the implementation from a user's perspective. The demonstrator provides a web application with an easy-to-use interface, which we present in this chapter with a step-by-step walk-through. The minimal prerequisites for a user solely include a browser and an account for the Austrian mobile phone signature (see subsection 2.1.4). Even though the application should be working in every state-of-the-art browser, we used Mozilla's Firefox¹ during development and can, therefore, recommend it.

1. If a user accesses the application at "importdemo.iaik.tugraz.at/SP/", the browser presents the eIDAS agent's initial page similar to Figure B.1. To start the derivation process, the user must simply click on the "NEXT" button.
2. The demonstrator creates a SAML request and displays it to the user in an encoded and decoded format as shown in Figure B.2. The request features a list of attributes that should be retrieved from the IdP. The user should continue with the default settings and click on the "SUBMIT" button to send the request from the eIDAS agent to the eIDAS node.

Anmeldung an: Sovrin Demo SP

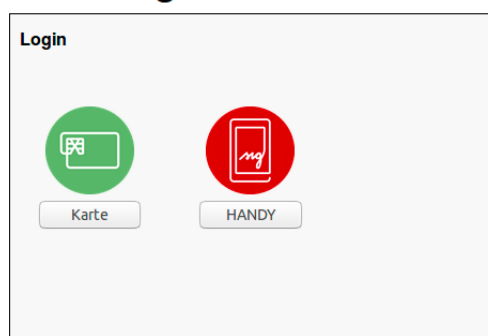


Figure 6.1.: Select authentication method

¹www.mozilla.org/en-US/firefox/.

6. Demonstrator

3. Subsequently, the application redirects the user to the eIDAS node. There the user must select an authentication method as depicted in Figure 6.1. In case the user has an Austrian mobile phone signature, the "HANDY" button below the red logo should be selected. The green "Karte" button serves testing purposes with a test-user.

Anmeldung an: Sovrin Demo SP

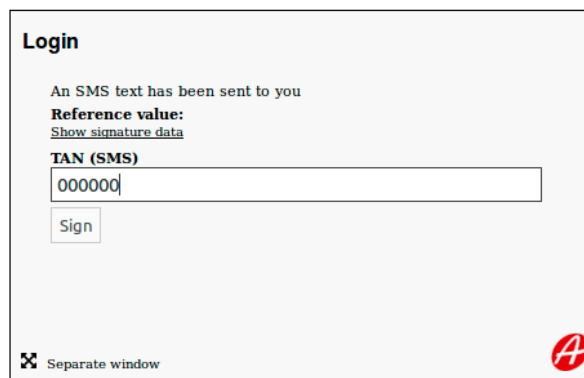


The screenshot shows a login window titled "Login". It contains two input fields: "Mobile phone number:" with the value "1234123456789" and a red eye icon to its right; and "Signature password:" with a masked password "*****". Below the password field is an "Identify" button. At the bottom left, there is a checkbox labeled "Separate window" which is checked. At the bottom right, there is a red circular logo with a white letter "A".

Figure 6.2.: Authentication field

4. Next, the user is redirected to the IdP. There, an input mask requests the phone number and the password for the mobile phone signature as shown in Figure 6.2. After entering the correct credentials and clicking on "Identify", the user must complete the second-factor authentication.

Anmeldung an: Sovrin Demo SP



The screenshot shows a login window titled "Login". It contains the text "An SMS text has been sent to you" followed by "Reference value:" and a link "Show signature data". Below this is a "TAN (SMS)" input field with the value "000000" and a "Sign" button. At the bottom left, there is a checkbox labeled "Separate window" which is checked. At the bottom right, there is a red circular logo with a white letter "A".

Figure 6.3.: Field to insert TAN from a short message

6. Demonstrator

- Depending on the second-factor authentication method that a user has chosen for the mobile phone signature, he either receives a six-digit TAN short message or must verify the fingerprint with an app on the phone. In both cases, the mobile phone is used for the second-factor authentication to increase the security. Figure 6.3 shows the input mask where the user should enter the TAN. After clicking on “Sign”, the TANs correctness is verified. Following this, the IdP issues the personal user information and transmits it via the eIDAS Node, and wrapped into a SAML assertion, back to the eIDAS agent.

PERSONAL DATA WITH CLAIM

Attribute	Values
http://eidas.europa.eu/attributes/naturalperson/CurrentFamilyName	[Mustermann]
http://eidas.europa.eu/attributes/naturalperson/CurrentGivenName	[Max]
http://eidas.europa.eu/attributes/naturalperson/DateOfBirth	[1940-01-01]
http://eidas.europa.eu/attributes/naturalperson/PersonIdentifier	[AT/SO/MiACFWU9RnPcnoTZzCXDb1Wlqv4=]
DID	5QoPvTNSAfDP8p9HWYNTQG
Public Key	3QRLhdimZwTNbaNSIRID8ZHBPpmtfvSj9AtByUFkriEd
Claim	<pre>{ "witness": null, "signature_correctness_proof": { "se": "151827072811684216645721641802430443442435048959", "c": "1798358498699703362217346341777421506527214267385" }, "rev_reg": null, "rev_reg_id": null, "signature": { "r_credential": null, "p_credential": { "m_2": "264604983415677358548049980233406979730730659", "a": "904919257314664173048110390421386021200436137214" } } }</pre>

Figure 6.4.: Summary after claim creation is complete

6. Demonstrator

6. The eIDAS node redirects the user back to the eIDAS agent where a screen like Figure B.3 is depicted. There, the browser displays the assertion in an encoded, an encrypted and a plain text version. With a click on "SUBMIT", the Sovrin part of the eIDAS agent starts the derivation. During this process, the agent creates and registers a DID, triggers the consensus protocol and creates a schema, a credential definition, and a claim.
7. After the process finished, the user is redirected to the last page that looks like Figure 6.4. There a table shows the name, date of birth and identifier of the SAML assertion along with the did, verification key, claim, credential definition and schema generated in the derivation process. The displayed claim and DID are for privacy reasons ephemeral. The server deletes all personal data, including private keys and wallets, from the SSI system immediately after the claim creation finished.

7. Discussion

In the following chapter, we discuss our concept (see chapter 4) and the PoC (see chapter 5). The first section presents properties that define our solution and a discussion on why they hold. In the subsequent section, other derivation approach candidates are discussed and compared to our concept.

7.1. Properties

The section below shows a list of properties that our PoC implements and which make our solution unique. Each property features a description of why it is essential and how we achieved it.

Trustworthiness: In the concept (see chapter 4) the nodes “substitute” the signature of the IdP, because of a non-compliant format. To keep the information trustworthy, each derived identity has a corresponding proof and multi-signature. The concept ensures that not a single agent or node can manipulate their creation.

The identity agent is not directly taking part in the proof creation process, it only forwards the assertion to the nodes and creates a claim including the multi-signature. The agent has only two options to interfere in the derivation process: Either by changing the assertion or by creating a claim with invalid data. In the first case, the nodes would identify the assertion as a fake, because the signature cannot be validated anymore, and subsequently abort the process. In the second case, the prover would uncover the claim as forged, because there does not exist a proof on the ledger that correlates with the attributes inside the claim.

The nodes create proofs and multi-signatures as a collective and by using the BFT protocol. This protocol ensures that a minority of malicious nodes cannot manipulate the proof creation process. We consider the process trustworthy because no single entity or minority can manipulate the derivation process.

7. Discussion

Performance and scalability: The PoC uses in comparison to Bitcoin¹ or Ethereum² a permissioned ledger, which means that only selected parties can host nodes and operate the ledger. Instead of a “proof-of-work-algorithm”, the concept takes advantage of the BFT protocol. The permissioned ledger reduces the need for “mining” and therefore, decreases the network’s computing power while increasing the system’s scalability. [2]

Decentralized verification: In our concept a local wallet stores the claims. Therefore, the verification can be handled locally without the need for a centralized third party. After receiving a claim, a verifying user can check that the identity agent issued the claim and that it has not been manipulated. Next, the user verifies the multi-signature and the proof, to ensure that the data originated from the given IdP. Due to the decentralized architecture of our PoC, these checks can happen locally, which reduces the workload for the IdM system. Further, the local checks increase the privacy because an IdP cannot correlate which parties exchanging identity information with each other.

7.2. Design Decisions

During the design of our concept, other solutions were under discussion. We shortly describe each approach and elaborate on why our final concept appears to be the most suitable approach for our research goal.

7.2.1. Assertion Inside the Claim

One approach takes the SAML assertion from the identity interface and stores it without conversion inside the newly created claim. A verifying user would check the claim first and subsequently extract the assertion from the attributes. The assertion would have been in XML in comparison to the JSON format of the claim. Therefore, each user client must implement a function that can verify a SAML assertion.

¹bitcoin.org/.

²www.ethereum.org.

7. Discussion

The following list describes the flaws in this concept:

- Our SAML assertions expire after a few minutes. With the assertion, the claim's authenticity expires as well. Therefore, a prover would need to go through the derivation process every time a claim is needed. Besides being highly impractical, this procedure would counter the idea of a decentralized identity management system.
- Every time the certificate of the original IdM system changes, all existing claims are instantly invalidated. Further, each user has to update the certificates in the local agent implementation.
- The identity agent is not only in charge of issuing claims, but also to ensure the authenticity of the assertion. This responsibility makes it to a single-point-of-failure and an attractive target for attackers.

In our concept, issues described above do not occur, because we do not store the SAML assertion inside the claim. Instead, the SSI nodes generate collectively a multi-signature that they store on the ledger. Instead of accumulating the trust in a single identity agent, the trust is distributed across the nodes.

7.2.2. Identity Interface issues Claims

Another approach followed the idea of making the claim creation as responsibility of the identity interface, which would be an eIDAS node (see section 2.1) in the PoC. The advantage of this option is the identity interface's capability of having access to the user's data and being able of creating trustworthy claims. Nevertheless, there is one major disadvantage regarding the implementation of this concept option: A running system designed for another purpose must be adapted. If we take the eIDAS network as an reference IdM system, then this concept would mean that an eIDAS node must implement the functionality of an Sovrin agent (see section 2.6). As a result, two non-related projects develop one system component, which could result in functions blocking each other and a high amount of maintenance effort every time one of the projects releases an update. In the eIDAS network, changes on the nodes would have to be implemented in several countries, which would cause a lot of implementation and maintenance overhead.

7. Discussion

Since our goal was to derive identity data from an existing IdM system, we considered subsequent changes on a system component in such an extent as noncompliant with our requirements. Our concept leaves the IdM untouched and creates a new component called the identity agent which we easily integrate into the modular structure of the SSI network.

7.2.3. Individual Revocation

In our concept, we propose a revocation mechanism, where the identity agent maintains a revocation registry, which the agent is updating regularly by sending attribute queries to the IdP. Another approach would instruct the user, who is verifying the claim, to check the claim's validity. This process would require the user to communicate directly with the IdP, which has the following disadvantages:

- It contrasts the idea of a decentralized SSI system if the revocation process includes a centralized solution.
- A curious IdP could analyze the revocation requests. Who tries to look up which identity entry? Which entries are the most requested ones? This behavior could harm the identity holder's privacy.
- The client software of indy (see section 2.6) would have to be adapted to enable a connection with the IdP because it is not designed to work with custom revocation solutions.

Our concept uses a revocation storage that preserves the user's privacy in a decentralized context. We introduced the identity agent as a broker, which links the SSI system with a traditional IdM system and therefore, ensures that both systems can be used without significant adoptions.

8. Security Analysis

In this chapter, we will analyze the security of the proposed concept and our implementation roughly based on the Common Criteria process, which follows the ISO 15408-1:2009 [7] standard. The analysis will specify a Target of Evaluation (TOE) along with the actors involved in the evaluation as well as assumptions we made. The following sections will focus on threats and threat agents that could potentially harm our systems. Finally, we will specify security objectives and describe how the countermeasures integrated into the system will ensure that the system is protected against the threats.

8.1. Target of Evaluation and Actors

The Target of Evaluation (TOE) specifies the scope for the security analysis. Since the concept aims to connect existing systems that can be exchangeable, the TOE focuses on the connecting components developed during this thesis. Therefore, the TOE includes the identity agent and the part of the SSI nodes and ledger used in the derivation process. The identity interface, the IdP and other parts of the SSI system are not in the focus of the TOE. Figure 8.1 depicts the TOE. The following list of actors specifies the components included in the evaluation:

User Client: The user starts the derivation process and has to authenticate herself against the IdP. The identity agent issues the claim for the user, who can subsequently utilize the user client to store the credentials and share it with other users. Especially the interaction with the identity agent will be essential for this evaluation. The user client is part of the TOE.

Identity Agent: The identity agent forms the central part of the concept. It coordinates the data flow between the identity interface and the SSI nodes and creates the claim for the user. These actions make the identity agent to a lucrative target for attackers. Therefore it is included in the TOE.

8. Security Analysis

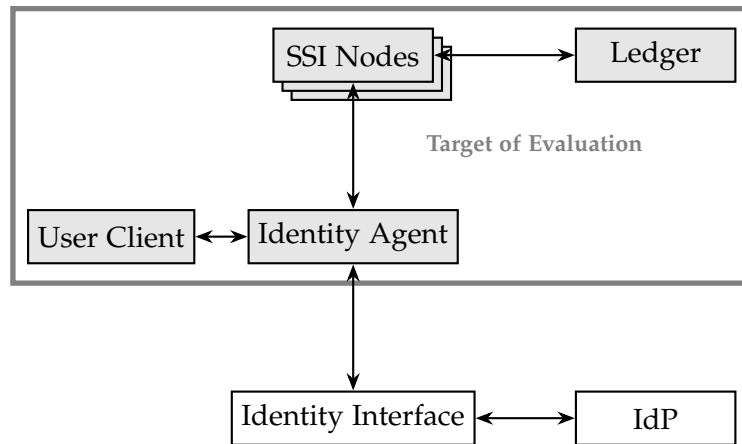


Figure 8.1.: Identity agent, SSI nodes and the ledger are part of the TOE

Identity Interface: The identity interface offers an interface for communication between the identity agent and the IdP. It must provide signed identity data to verify the origin of the message. Since we do not define which identity system should be used, we do not include the identity interface into our TOE.

IdP: The IdP stores and maintains the user's eID, from which the claim derives. The IdP should implement strong authentication and must be connected with the identity interface that could communicate with the identity agent. Since our concept does not focus on a specific identity system, we exclude the IdP from the TOE.

SSI Node: The SSI nodes are responsible for request handling. Since they decide what to write on the ledger, they have to check every request carefully. Nodes are the entities with the highest trust in the SSI system. Therefore, they oversee the derivation process, which makes them tempting targets for attackers. Since different SSI systems are built on different architectures with distinct features, only the derivation process is part of the TOE.

Ledger: The SSI nodes operate the ledger, which stores among others DIDs, schemas and the proof for derivations. Even though an attacker would need to take over the nodes to control the ledger, the ledger plays an essential role in this evaluation, because it forms the backbone of trust in the system. Without the ledger, everybody could claim to be anybody, which would destroy the purpose of a decentralized identity system. Due to this, the ledger is part of the TOE.

8.2. Assumptions

The following section describes assumptions we use in our security evaluation. Based on them, we created attack-scenarios and threats. On the one hand, we define trust assumptions for the server-side actors from section 8.1. On the other hand, we specify general assumptions about the setup of the system.

The following enumeration shows the actor specific assumptions.

- AS.1 **User client is malicious:** The user client triggers an identity derivation, stores user credentials and uses claims to proof the identity. An attacker could try to impersonate as another user by using forged or stolen claims. We assume the user client to be malicious.
- AS.2 **Identity agent is honest but curious:** The identity agent offers due to the high amount of processed sensitive data a lucrative target for an attacker. A careful attacker gains the most if it stays disguised after taking over the identity agent and continuing the protocols while making little changes to satisfy its curiosity. Therefore, the identity agent is considered, to be honest, but curious.
- AS.3 **SSI nodes are possibly malicious:** The nodes are the most trusted entities in the SSI system and operate the ledger. It is highly likely that attackers could extensively try to take over one or more nodes, but it is improbable that all nodes are concurrently in control of an attacker. Due to this, the security analysis assumes that a small number of nodes could be malicious.
- AS.4 **Identity interface is trustworthy:** An external system provides the interface. We assume that the user only stores personal data in trustworthy systems.
- AS.5 **IdP is trustworthy:** The IdP stores the user's original identity data. Like the identity interface, our PoC uses the IdP from an external party, which we consider in this security analysis as trustworthy.

The following list describes general assumptions.

- AS.6 **Correct implementation:** We assume a bug-free implementation of the concept without security flaws.

8. Security Analysis

- AS.7 **Correct setup:** The analysis assumes that all components are set up correctly. This setup includes the key generation and the initial exchanges of certificates and public keys.
- AS.8 **Secure environment:** We assume that the program execution happens in a safe environment. This environment includes restricted access, machines without malicious software on it, a secure physical location and a secure network connection.
- AS.9 **Availability:** The analysis assumes all components and resources to be available during the derivation process.
- AS.10 **Enough resources:** We assume that the machines used, have enough processing power and enough storage to deal with the identity derivation.

8.3. Assets

The security analysis defines assets, which hold valuable information for actors. Due to their value, assets are targets of attackers, and therefore, they are included in the TOE. The following enumeration shows a list of assets in the identity derivation system.

- A.1 **Personal data:** Personal data are retrieved from the IdP, validated by the nodes and packed into a claim by the identity agent. The data include sensitive information about the user like name and date of birth.
- A.2 **Private keys:** The identity interface, identity agent, the SSI nodes and the user utilize private keys to create signatures because asymmetric cryptography protects the integrity and authenticity of messages. The cryptographic material is highly sensitive and therefore, must be protected.
- A.3 **Claims:** Claims are personalized and individually issued data structures that resemble identification documents. Like someone would try to stay in control over personal documents in real life to prevent identity theft and fraud it is crucial to protect their virtual equivalent as well.

8. Security Analysis

A.4 **Ledger:** The ledger keeps a history of the submitted transactions. It registers every successful derivation and therefore, serves as a counterpart for authenticity checks of issued claims with derived data. The ledger further holds information about DID documents and their public keys, which play an essential role in establishing decentralized communication channels and verifying claims. Due to this, it is crucial to protect the ledger.

8.4. Threat Agents

The following section describes the threat agents in this security analysis. They act against the assets described above. We introduce agents that threat actors and their assets, as well as agents potentially harming the communication between the actors.

- TA.1 **Identity Agent Attacker:** In the security assumptions we consider the identity agent honest but curious. It retrieves and forwards identity assertions and creates claims. We expect the agent to generally follow the protocol and therefore, to ensure a correct data flow between the environments due to its honesty. Since the identity agent is also curious, it may alter messages to its benefit. If these changes remain undetected, they could lead to identity documents with faked personal data.
- TA.2 **Client Attacker:** Even though, the user is considered to be trustworthy, attacks on her client software pose a serious threat. A decentralized environment profits from distributed storage, but could also bring the local software into the focus of attackers. A threat agent could be tempted to use stolen data for impersonation or other privacy harming activities.
- TA.3 **Node Attacker:** The nodes operate the ledger, process all requests and replace the identity assertion's signature with their multi-signature. All of these tasks require a significant amount of responsibility and trust. In our assumption we considered the nodes to be possibly malicious. A malicious node could actively try to change the ledger, hold back messages, manipulate the electoral outcome and spread misinformation.
- TA.4 **Network Attacker:** A network attacker would abuse the necessity to communicate over a network in a decentralized environment extensively. Messages include among others sensitive topics like DID discovery, DID creation, claim creation and claim proofing. A network attacker would very likely seize the role of a

8. Security Analysis

Man-In-The-Middle. This attack means that this threat agent would sniff in the traffic and try to read and alter messages, which entities in the network exchange with each other.

8.5. Threats

This section describes the threats in this security analysis. A threat describes an action executed by a threat agent, who aims to harm an asset. Therefore, the following list describes each threat and its relation to agents and assets. We further visualized the relationship in Table 8.1.

- T.1 Identity agent changes assertion:** The identity agent receives an identity assertion from the identity interface containing personal user attributes. An identity agent attacker could try to change the assertion in his favor by changing, for example, the date of birth or the user's name. If the nodes accept this assertion, a claim with faulty data is created that is nevertheless verifiable. An honest proofing entity would unknowingly accept the faked claim. These would be the real-life equivalent of a passport issued by the authorities with an incorrect name or date of birth. A police officer would consider the passport to be legit and trust the information written there.
Harmed asset: A.1 personal data
Executing threat agent: TA.1 identity agent attacker, TA.3 node attacker
- T.2 Expired assertion:** An identity agent attacker could try a replay attack with an old assertion. In this attack, the agent would store a previously received assertion and try to resend it to the nodes at a later point in time to create an identity, which could feature expired or revoked attributes.
Harmed asset: A.1 personal data
Executing threat agent: TA.1 identity agent attacker
- T.3 Self signed claim:** An client attacker could create its unique credential definition and issue non-legit claims. These claims are cryptographically verifiable and could be mistaken for official claims issued by the identity agent.
Harmed asset: A.3 claim
Executing threat agent: TA.2 client attacker

8. Security Analysis

- T.4 **Steal DID:** A client attacker could try to get the hands on the user's private keys. In this case, the threat agent could impersonate as the user using the DID structure.
Harmed asset: A.2 private keys
Executing threat agent: TA.2 client attacker
- T.5 **Steal claim:** A claim stores personal attributes and provides a cryptographic structure for third-party verification. If a client attacker steals a user's claim, the threat agent could take advantage of this information to impersonate as the user.
Harmed asset: A.3 claim
Executing threat agent: TA.2 client attacker
- T.6 **Delete ledger:** The assumptions consider the nodes to be possibly malicious. If this is the case, a node attacker could try to delete the ledger partly or as a whole to get rid of individual entries. These actions could range from deleting specific DIDs or credentials to making the whole identity system unusable by deleting the entire ledger.
Harmed asset: A.4 ledger
Executing threat agent: TA.3 node attacker
- T.7 **Add a malicious entry to ledger:** A node attacker could use its capabilities to append new transactions to the ledger. These entries could include fake DIDs, claims and derivation proofs. An unaware user could eventually trust the forged transactions.
Harmed asset: A.4 ledger
Executing threat agent: TA.3 node attacker
- T.8 **Manipulate derivation proof:** The nodes jointly create proofs for derivation based on the request and identity assertion from the identity agent. A node attacker could try to manipulate the proof creation process. This action includes changing the request's properties or dropping or postponing a package. These actions could lead to legit requests being rejected or faulty transactions being written to the ledger.
Harmed asset: A.1 personal data
Executing threat agent: TA.3 node attacker

8. Security Analysis

T.9 Attack network traffic: In a decentralized system user clients communicate with agents like the identity agent; agents communicate with nodes and external components like the identity interface and nodes communicate with each other. An network attacker could interfere in the network traffic and try to extract or alter user information.

Harmed asset: A.1 personal data, A.3 claim

Executing threat agent: TA.4 network attacker

	A.1 Personal data	A.2 Private keys	A.3 Claims	A.4 Ledger	TA.1 Identity Agent Attacker	TA.2 Client Attacker	TA.3 Node Attacker	TA.4 Network Attacker
T.1 Identity agent changes assertion	✓				✓			
T.2 Expired assertion	✓				✓		✓	
T.3 Self signed claim			✓			✓		
T.4 Steal DID		✓				✓		
T.5 Steal caim			✓			✓		
T.6 Delete ledger				✓				✓
T.7 Add a malicious entry to ledger				✓				✓
T.8 Manipulate derivation proof	✓						✓	
T.9 Attack network traffic	✓		✓					✓

Table 8.1.: Shows the relation between threats, assets and threat agents

8.6. Security Objectives

The security objectives describe strategies to mitigate the threats listed in the section before. They should define a general mitigation mechanism instead of a concrete implementation. Each security objective states, which threat it counters. This relationship is visualized in Table 8.2.

8. Security Analysis

O.1 **Ensure assertions are signed:** Messages containing personal data must be signed. Signatures ensure authenticity and allow an honest proofer to ensure the correct origin of the data. All components dealing with sensitive data should verify the signatures appended to assertions.

Covered Threats: T.1 identity agent changes assertion

O.2 **Ensure expiration checks:** The TOE ensures that assertions feature an expiration date. This mechanism protects against replay attacks if all honest parties reliably execute checks.

Covered Threats: T.2 expired assertion

O.3 **Known origin:** Parties involved in the credential checking like verifiers must be able to determine and verify the correct origin of a credential. A real-world analogy for this problem would include the issuing of a degree. A future employer would not trust the degree of a fictional university of an applicant. Therefore, the TOE must ensure that a verifier can know which issuers are allowed to issue a particular type of credentials and how to verify the authenticity.

Covered Threats: T.3 self signed threat

O.4 **Protect DIDs:** Sensitive data must be stored securely to prevent attackers from stealing private keys. Besides the storage, the key material should be created in a safe environment with well-established algorithms.

Covered Threats: T.4 steal DID

O.5 **Protect claims:** The TOE protects the claims from theft. It further ensures that only the rightful owner can use the credential information to confirm the identity.

Covered Threats: T.5 steal claim

O.6 **Indelible ledger:** The TOE ensures that the ledger can neither be deleted as a whole nor can single entries be removed or changed. It implements a write-only data structure, where only new entries can be appended.

Covered Threats: T.6 delete ledger

O.7 **Distributed consensus strategy:** The TOE guarantees that no single entity may write new data on the ledger. Only nodes can write data on the ledger if a majority of them votes for it. The voting process should be transparent and comprehensible.

Covered Threats: T.7 add a malicious entry to ledger

8. Security Analysis

	O.1 Ensure assertions are signed	O.2 Ensure expiration checks	O.3 Known origin	O.4 protect DIDs	O.5 Protect claims	O.6 Indelible ledger	O.7 Distributed consensus strategy	O.8 Trustworthy derivation proof	O.9 Protect network traffic
T.1 Identity agent changes assertion	✓								
T.2 Expired assertion		✓							
T.3 Self signed claim			✓						
T.4 Steal DID				✓					
T.5 Steal claim					✓				
T.6 Delete ledger						✓			
T.7 Add a malicious entry to ledger							✓		
T.8 Manipulate derivation proof								✓	
T.9 Attack network traffic									✓

Table 8.2.: Shows the relation between threats and security objectives.

O.8 Trustworthy derivation proof: The TOE protects the proof creation process. Reliable nodes should identify malicious requests and omit them, to ensure that only unaltered data ends up on the ledger.

Covered Threats: T.8 manipulate derivation proof

O.9 Protect network traffic: The TOE ensures encrypted traffic between the components. State-of-the-art transport protocols should be in place while agents, nodes and external entities communicate with each other.

Covered Threats: T.9 attack network traffic

8.7. Countermeasures

The countermeasures are concrete implementations of the security objectives in our PoC. They show what we have done to mitigate the threats given in this security analysis. The following list describes the countermeasures in detail. Due to the relation with the objective, the same name and enumeration of the list items are in place.

- O.1 **Ensure assertions are signed:** In our PoC the identity interface signs SAML assertions using XMLDSIG. All components dealing with the assertions, namely the identity agent and the SSI nodes are obligated to verify the signature and the origin once they receive an assertion. If the signature is not verifiable, or the message does not include a signature, an honest component aborts the operation to ensure that only unaltered information finds its way into the verifiable claim.
- O.2 **Ensure expiration checks:** Like the signature, the SAML assertion from the identity agent must feature an attribute with an expiration date. The identity agent and the SSI nodes check the expiration date every time they receive an assertion. If a message with an expired assertion arrives, the components assume a replay attack and abort the operation.
- O.3 **Known origin:** Since the identity agent is not a private person, it does not pose a privacy problem to make the agent's DID publicly available. A potential verifier could look up the DID beforehand and request the verification key and credential definition from the nodes. Claims are based on a specific credential definition and always signed using the issuer's DID. Due to this, a verifier knows a claim's origin.
- O.4 **Protect DIDs:** We used the indy-crypto library and elliptic curves to generate a private key. Since our PoC generates DIDs for demonstration purposes, which are not linked to real-world user wallets, we delete the keys after the claim creation. Therefore, no persistent secure storage is needed. Due to the current development of the Indy project, we assume that in the future a connection to secure storage for persistent claims will be available.
- O.5 **Protect claims:** Claims are tied to a specific DID. Even if an attacker steals a claim from another client, she would not be able to impersonate as this user due to the missing private keys. The architecture of DIDs and verifiable claims implemented in indy, avert this threat.

8. Security Analysis

- O.6 **Indelible ledger:** Ledgers are per definition write-only. Indy implements a Merkle Tree using SHA-1 hashes to ensure that existing information cannot be deleted or changed. Since different nodes store a copy of the ledger, the deletion of one copy from one node would not “delete” the collective knowledge.
- O.7 **Distributed consensus strategy:** The PoC uses an extended version of the plenum protocol. Indy takes advantage of this consensus protocol every time a transaction should be written to the ledger and therefore, conceptually ensures that there is not any other way to extend the ledger. Further, plenum guarantees that new transactions are only appended if more than two-third of the nodes approve with the correctness of the request.
- O.8 **Trustworthy derivation proof:** For the derivation process our extended version of plenum is used. We take advantage of the protocol’s capabilities including the different phases and the validity checks. The derivation proof is as safe as the remaining SSI system, under the following assumption: More than two-thirds of the nodes act trustworthy. If this is the case, the consensus protocol guards the proof creation process, the distributed ledger architecture protects the storage and the indy-crypto library secures the verification.
- O.9 **Protect network traffic:** The network uses the ZeroMQ framework for exchanging messages between agents and nodes. The framework aims at distributed systems and supports elliptic curve encryption for network traffic to prevent curious middleman from listening.

8.8. Conclusion

In this security analysis, we took a look at the valuable assets in our system and how threat agents could harm them. We showed how to avert these threats theoretically in the security objectives. Finally, the countermeasures describe how the PoC fulfills every security objectives. Therefore, given that our assumptions hold, we conclude that the PoC can be considered secure.

9. Future work

The concept of this thesis opened a new approach in the field of identity derivation, and we are looking for further development in this field as SSI systems seem to be a new and promising research field for IdMs [2]. Potential lies on the one hand in the innovative power of further privacy protecting developments in connection with new platforms and cryptographic methods. On the other hand, we see possibilities for future work on the adaption of our proposed concept for fields outside of identity management. The following list presents a selection possibilities for further research.

Privacy preserving mechanisms: For the derivation proof creation, the nodes currently receive the identity assertion and have to read the attributes to verify the signature's correctness. Though we basically expect the nodes to be trustworthy, in a worst-case scenario, an intruded node could potentially get access to sensitive data. The introduction of blind signatures[51] could help to improve the privacy further.

Anonymous revocation: We implemented a revocation mechanism that regularly checks if the original identity data from the IdP have changed. Currently, the identity agent is responsible for maintaining the revocation storage and therefore, must store sensitive information about the claims (see section 4.1). For this thesis, it was a crucial requirement that the existing identity system remains unaltered. If future work would omit this requirement, an interface could be created that allows the identity agent to retrieve blinded updates for the revocation list. This approach would remove the burden of storing user data at the identity agent.

Secure storage and key recovery: With the growing interest in decentralized systems, users' wallets store more sensitive data. Accordingly, the need for better protection rises. Even though a controlled environment can guarantee high security, history has shown that once the masses reach out for new technologies, developers should reconsider usability and security features. These aspects include the question for a secure storage: When is a device secure enough to store sensitive information? Should key material be stored on a portable device? What happens if a key is lost? How can the owner recover a key, but not an attacker? Questions like these show that there is still much potential research for securing the users' data.

9. Future work

Network analysis: In decentralized systems, users are communicating with agents, who communicate with nodes, which further exchange messages with each other. Especially during the execution of the consensus protocol several messages and acknowledgments are transported between the nodes. A scientific analysis of the network traffic could potentially help to increase as well efficiency as security of the network and the protocol.

Distributed trust: Our identity derivation features the extended RBFT protocol that substitutes the signature from the original message with a new multi-signature created by distributed nodes (see section 5.3). This solution is not limited for the use in IdMs and could be adapted for other areas where the authenticity of data are the primary concern like in insurance and finance businesses.

10. Conclusion

In the course of this thesis, we created a concept for an identity derivation process. An identity owner may import personal attributes from an IdM to an SSI system. A user of the proposed system can derive an existing eID and utilize it for direct communication with third-party-services in a privacy-preserving manner. Since the full migration of an IdM system in operation to a new identity model provides a source of errors, a primary requirement for the concept demanded to keep the existing IdM system unaltered. Therefore, we introduced an identity agent, which provides an interface between the traditional and the SSI system. The identity agent further issues verifiable claims that include eID attributes and maintains a revocation list that ensures an up-to-date status of the personal information. The concept further described trusted nodes that utilize an extended Byzantine fault tolerance protocol to create a derivation proof, without a trusted third-party.

We further proofed the feasibility of our concept with the implementation of a PoC. It utilized the eIDAS network with access to the Austrian mobile phone signature as a source for qualified eIDs. The eID has the form of a SAML assertion that the validator nodes and the identity agent transform into a verifiable claim in JSON format. The nodes further create a multi-signature and a proof that guarantees the origin of the information from the eIDAS IdP. A distributed ledger stores the proof to make it available for the public. Therefore, a verifying user could trace the data's authenticity.

Moreover, this thesis includes a demonstrator that allows a user with an Austrian mobile phone signature to explore the identity derivation. A web application guides the user with a few clicks through the process. The demonstrator showed the feasibility in a real-world scenario and did not require a prior setup.

10. Conclusion

Furthermore, this thesis concluded with a security analysis, which was roughly based on the Common Criteria process. It described the actors and assets, as well as the attackers and threats they constitute. The analysis resulted in the conclusion that the proposed countermeasures in concept and implementation prevent attackers from influencing the derivation process and faking identities.

In summary, we showed with a concept, an implementation, and a security analysis a feasible way to derive eIDs from an existing IdM system into an SSI system, without a central trusted third-party. This approach improves state-of-the-art IdM systems by giving identity owners more privacy while ensuring easy integration without changing existing systems.

Appendices

Appendix A.

Listings

This section contains listings from the chapter 5.

```
{
  "@context": [
    "https://w3id.org/identity/v1",
    "https://w3id.org/security/v1"],
  "id": "http://example.gov/credentials/3732",
  "type": ["Credential", "PassportCredential"],
  "name": "Passport",
  "issuer": "https://example.gov",
  "issued": "2010-01-01",
  "claim": {
    "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
    "name": "Alice Bobman",
    "birthDate": "1985-12-14",
    "gender": "female",
    "nationality": {
      "name": "United States"},
    "address": {
      "type": "PostalAddress",
      "addressStreet": "372 Sumter Lane",
      "addressLocality": "Blackrock",
      "addressRegion": "Nevada",
      "postalCode": "23784",
      "addressCountry": "US"},
    "passport": {
      "type": "Passport",
      "name": "United States Passport",
      "documentId": "123-45-6789",
      "issuer": "https://example.gov",
      "issued": "2010-01-07T01:02:03Z",
      "expires": "2020-01-07T01:02:03Z"} },
  "signature": {
    "type": "LinkedDataSignature2015",
    "created": "2016-06-21T03:40:19Z",
    "creator": "https://example.com/jdoe/keys/1",
    "domain": "json-ld.org",
    "nonce": "783b4dfa",
    "signatureValue": "Rxj7Kb/tDbGHFAs6dd...ibsNk="}
}
```

Listing A.1: Example of a verifiable claim and its properties from [18] in JSON-LD format.

Appendix A. Listings

```
{
  "schema_id": "Th7MpTaRZVRyNpiabds81Y:2:eidasimport:1.0",
  "cred_def_id": "AgQbxDbdWX9KFRscaYrT4C:3:CL:Th7MpTaRZVRyNpiabds81Y:2:eidasimport:1.0",
  "rev_reg_id": null,
  "values": {
    "familyName": {
      "raw": "Mustermann",
      "encoded": "7711711511610111410997110110"
    },
    "proof": {
      "raw": "{
        'state_proof': {
          'root_hash': 'Ar88f15dWUkchrXkYXQKJbYF6hvCttAKgVgjYRSVRgj',
          'proof_nodes': '+Lf4tbbhBIGV1ZDQ1NmY3YTEwZ...
            JiNzZiNzVhZmRiMzkyZDRiY2Fh0GYifX0=',
          'multi_signature': {
            'signature': 'RXRfTLzkiJBDBaPbtB63KkK9D...
              rkDLYc9gvffGGLdRKMqgZWBjeuavKg7R6X ',
            'value': {
              'pool_state_root_hash': 'fz9sZMnQhr3A78Q2vWP3hD2PfC8RFmyQu8u3hoXSF2M
                ',
              'ledger_id': 42,
              'timestamp': 1541670621,
              'txn_root_hash': 'ETVf3GUfkd8CY7BNu8iELpCaJ742HmsZkjABWGusa9p6 ',
              'state_root_hash': 'Ar88f15dWUkchrXkYXQKJbYF6hvCttAKgVgjYRSVRgj'
            },
            'participants': ['Gamma', 'Delta', 'Alpha']
          }
        },
        'seqNo': 1,
        'txnTime': 1541670621,
        'reqId': 49550,
        'data': {'hash': 'eed456f7a10fd942c256...ae87ff2b76b75afdb392d4bcaa8f'},
        'identifier': 'MSjKTWkPLtYoPEaTF1TUDb',
        'type': '42421'
      }",
      "encoded": "1233911511697116101951121...1210139583239525052504939125"
    },
    "dataOfBirth": {
      "raw": "1940-01-01",
      "encoded": "49575248454849454849"
    },
    "eidasloa": {
      "raw": "http://eidas.europa.eu/LoA/high",
      "encoded": "10411611611258474710110510...0111747761116547104105103104"
    },
    "hash": {
      "raw": "eed456f7a10fd942c256644330...87ff2b76b75afdb392d4bcaa8f",
      "encoded": "10110110052535410255974948...0098515750100529899979756102"
    },
    "personIdentifiertype": {
      "raw": "AT/SO/MiACFWU9RnPcnoTZzCXDb1Wlgv4=",
      "encoded": "65844783794777105656770878...1226788689849871081031185261"
    },
    "givenName": {
      "raw": "Max",
      "encoded": "7797120"
    }
  }
}
```

Appendix A. Listings

```
},
"signature": {
  "p_credential": {
    "m_2": "7898269061972268921912382042...6215171490096041510257954674",
    "a": "8090797822958942791386...5709882669446437190683",
    "e": "25934472305506205990702549148...388009376199277",
    "v": "69889728276297184069396985649...895104834406337"
  },
  "r_credential": null
},
"signature_correctness_proof": {
  "se": "782290705311898700449117571...08723924946726183",
  "c": "3306722718986732828804721883...19129208579242323"
},
"rev_reg": null,
"witness": null
}
```

Listing A.2: Claim from eIDAS agent for a user

```
<?xml version="1.0" encoding="UTF-8"?>
<saml2p:AuthnRequest xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol" xmlns:ds
="http://www.w3.org/2000/09/xmldsig#" xmlns:oidas="http://oidas.europa.eu/saml
-extensions" xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion" Consent="urn:
oasis:names:tc:SAML:2.0:consent:unspecified" Destination="https://vidp.gv.at/
eIDAS_node/oidas/ColleagueRequest" ForceAuthn="true" ID=
_X_3DNfpZUwOpCRy_wA0Yf17e.lDf6pnBEpcvz13w1m0cHB2lkSHHbL.YbdMoVS" IsPassive="
false" IssueInstant="2018-11-09T08:40:23.546Z" ProviderName="DEMO-SP-CA"
Version="2.0">
<saml2:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">http://
importdemo.iaik.tugraz.at/SP/metadata</saml2:Issuer>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n
#"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-
sha512"/>
    <ds:Reference URI="#_X_3DNfpZUwOpCRy_wA0Yf17e.lDf6pnBEpcvz13w1m0cHB2lkSHHbL
.YbdMoVS">
      <ds:Transforms>
        <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"/>
        <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      </ds:Transforms>
      <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha512"/>
      <ds:DigestValue>oZUEqWI...pu1Aw5CMX32YnEQ==</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>dofi1LbSlc7mT2yTE9I...SbHF7iqEzqSgE=</ds:SignatureValue>
  <ds:KeyInfo>
    <ds:X509Data>
      <ds:X509Certificate>MIIFQTCCAykCBFTI...gcYvyW8H0vSLV1AzUFuCCe=</ds:
X509Certificate>
    </ds:X509Data>
  </ds:KeyInfo>
</ds:Signature>
<saml2p:Extensions>
  <oidas:SPTYPE>public</oidas:SPTYPE>
```

Appendix A. Listings

```
<eidas:RequestedAttributes>
  <eidas:RequestedAttribute FriendlyName="D-2012-17-EUIdentifier" Name="http
    ://eidas.europa.eu/attributes/legalperson/D-2012-17-EUIdentifier"
    NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri" isRequired
    ="false"/>
  <eidas:RequestedAttribute FriendlyName="EORI" Name="http://eidas.europa.eu/
    attributes/legalperson/EORI" NameFormat="urn:oasis:names:tc:SAML:2.0:
    attrname-format:uri" isRequired="false"/>
  <eidas:RequestedAttribute FriendlyName="LEI" Name="http://eidas.europa.eu/
    attributes/legalperson/LEI" NameFormat="urn:oasis:names:tc:SAML:2.0:
    attrname-format:uri" isRequired="false"/>
  <eidas:RequestedAttribute FriendlyName="LegalAdditionalAttribute" Name="http
    ://eidas.europa.eu/attributes/legalperson/LegalAdditionalAttribute"
    NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri" isRequired
    ="false"/>
  <eidas:RequestedAttribute FriendlyName="SEED" Name="http://eidas.europa.eu/
    attributes/legalperson/SEED" NameFormat="urn:oasis:names:tc:SAML:2.0:
    attrname-format:uri" isRequired="false"/>
  <eidas:RequestedAttribute FriendlyName="SIC" Name="http://eidas.europa.eu/
    attributes/legalperson/SIC" NameFormat="urn:oasis:names:tc:SAML:2.0:
    attrname-format:uri" isRequired="false"/>
  <eidas:RequestedAttribute FriendlyName="TaxReference" Name="http://eidas.
    europa.eu/attributes/legalperson/TaxReference" NameFormat="urn:oasis:
    names:tc:SAML:2.0:attrname-format:uri" isRequired="false"/>
  <eidas:RequestedAttribute FriendlyName="VATRegistration" Name="http://eidas.
    europa.eu/attributes/legalperson/VATRegistrationNumber" NameFormat="urn:
    oasis:names:tc:SAML:2.0:attrname-format:uri" isRequired="false"/>
  <eidas:RequestedAttribute FriendlyName="AdditionalAttribute" Name="http://
    eidas.europa.eu/attributes/naturalperson/AdditionalAttribute" NameFormat
    ="urn:oasis:names:tc:SAML:2.0:attrname-format:uri" isRequired="false"/>
  <eidas:RequestedAttribute FriendlyName="BirthName" Name="http://eidas.europa
    .eu/attributes/naturalperson/BirthName" NameFormat="urn:oasis:names:tc:
    SAML:2.0:attrname-format:uri" isRequired="false"/>
  <eidas:RequestedAttribute FriendlyName="CurrentAddress" Name="http://eidas.
    europa.eu/attributes/naturalperson/CurrentAddress" NameFormat="urn:oasis
    :names:tc:SAML:2.0:attrname-format:uri" isRequired="false"/>
  <eidas:RequestedAttribute FriendlyName="FamilyName" Name="http://eidas.
    europa.eu/attributes/naturalperson/CurrentFamilyName" NameFormat="urn:
    oasis:names:tc:SAML:2.0:attrname-format:uri" isRequired="true"/>
  <eidas:RequestedAttribute FriendlyName="FirstName" Name="http://eidas.europa
    .eu/attributes/naturalperson/CurrentGivenName" NameFormat="urn:oasis:
    names:tc:SAML:2.0:attrname-format:uri" isRequired="true"/>
  <eidas:RequestedAttribute FriendlyName="DateOfBirth" Name="http://eidas.
    europa.eu/attributes/naturalperson/DateOfBirth" NameFormat="urn:oasis:
    names:tc:SAML:2.0:attrname-format:uri" isRequired="true"/>
  <eidas:RequestedAttribute FriendlyName="Gender" Name="http://eidas.europa.eu
    /attributes/naturalperson/Gender" NameFormat="urn:oasis:names:tc:SAML
    :2.0:attrname-format:uri" isRequired="false"/>
  <eidas:RequestedAttribute FriendlyName="PersonIdentifier" Name="http://eidas
    .europa.eu/attributes/naturalperson/PersonIdentifier" NameFormat="urn:
    oasis:names:tc:SAML:2.0:attrname-format:uri" isRequired="true"/>
  <eidas:RequestedAttribute FriendlyName="PlaceOfBirth" Name="http://eidas.
    europa.eu/attributes/naturalperson/PlaceOfBirth" NameFormat="urn:oasis:
    names:tc:SAML:2.0:attrname-format:uri" isRequired="false"/>
</eidas:RequestedAttributes>
</saml2p:Extensions>
<saml2p:NameIDPolicy AllowCreate="true" Format="urn:oasis:names:tc:SAML:1.1:
  nameid-format:unspecified"/>
```

Appendix A. Listings

```
<saml2p:RequestedAuthnContext Comparison="minimum">
  <saml2:AuthnContextClassRef>http://eidas.europa.eu/LoA/high</saml2:
    AuthnContextClassRef>
</saml2p:RequestedAuthnContext>
</saml2p:AuthnRequest>
```

Listing A.3: An eIDAS SAML request from eIDAS agent

```
<?xml version="1.0" encoding="UTF-8"?>
<saml2:Assertion xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion" xmlns:eidas-
  natural="http://eidas.europa.eu/attributes/naturalperson" ID="_D_iDmIDFN.
  mWzUNBZ5TjR525UE0xF9RNblxQ_n2TDMY-0mgfxoRfYzQK7X5hjcR" IssueInstant
  ="2018-11-08T09:25:06.654Z" Version="2.0">
  <saml2:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">https://
    vidp.gv.at/eIDAS_node/eidas/metadata</saml2:Issuer>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
      <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n
        #"/>
      <ds:SignatureMethod Algorithm="http://www.w3.org/2007/05/xmldsig-more#sha256
        -rsa-MGF1"/>
      <ds:Reference URI="#_D_iDmIDFN.mWzUNBZ5TjR525UE0xF9RNblxQ_n2TDMY-
        0mgfxoRfYzQK7X5hjcR">
        <ds:Transforms>
          <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
            signature"/>
          <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
            <ec:InclusiveNamespaces xmlns:ec="http://www.w3.org/2001/10/xml-exc-
              c14n#" PrefixList="eidas-natural"/>
          </ds:Transform>
        </ds:Transforms>
        <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmenc#sha256"/>
        <ds:DigestValue>snKbhSDVEZV0Y1eMIV9rYsddrKMddrMeVv0hq2GP6QY=</ds:
          DigestValue>
        </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>KbGU+kYFOIoZf2rkq0VBrEU73I/J2...iRpPQAqmGlsHe5awBXLW4zeX7
        +9</ds:SignatureValue>
    <ds:KeyInfo>
      <ds:X509Data>
        <ds:X509Certificate>MIIEBzCCAm8CBFmjz7AwD...lr8l9mUro</ds:X509Certificate>
      </ds:X509Data>
    </ds:KeyInfo>
  </ds:Signature>
  <saml2:Subject>
    <saml2:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified"
      NameQualifier="http://C-PEPS.gov.xx">AT/SO/MiACFWU9RnRncnoTZzCXDb1Wlgv4=</
      saml2:NameID>
    <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
      <saml2:SubjectConfirmationData Address="129.27.142.188" InResponseTo="
        _5p6Z1kGp-9o0ZFNzBQGmFx_ASsnF-TKil-RuQFN5dAFd7HxBN6QFbr8HYHU.Nwi"
        NotOnOrAfter="2018-11-08T09:30:06.654Z" Recipient="http://importdemo.
        iaik.tugraz.at/SP/ReturnPage"/>
    </saml2:SubjectConfirmation>
  </saml2:Subject>
  <saml2:Conditions NotBefore="2018-11-08T09:25:06.654Z" NotOnOrAfter="2018-11-08
    T09:30:06.654Z">
    <saml2:AudienceRestriction>
```

Appendix A. Listings

```
<saml2:Audience>http://importdemo.iaik.tugraz.at/SP/metadata</saml2:Audience
>
</saml2:AudienceRestriction>
<saml2:OneTimeUse/>
</saml2:Conditions>
<saml2:AuthnStatement AuthnInstant="2018-11-08T09:25:06.654Z">
  <saml2:AuthnContext>
    <saml2:AuthnContextClassRef>http://eidas.europa.eu/LoA/high</saml2:
      AuthnContextClassRef>
    <saml2:AuthnContextDecl/>
  </saml2:AuthnContext>
</saml2:AuthnStatement>
<saml2:AttributeStatement>
  <saml2:Attribute FriendlyName="FamilyName" Name="http://eidas.europa.eu/
    attributes/naturalperson/CurrentFamilyName" NameFormat="urn:oasis:names:tc:
      SAML:2.0:attrname-format:uri">
    <saml2:AttributeValue xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:type="eidas-natural:CurrentFamilyNameType">Mustermann</saml2:
        AttributeValue>
  </saml2:Attribute>
  <saml2:Attribute FriendlyName="FirstName" Name="http://eidas.europa.eu/
    attributes/naturalperson/CurrentGivenName" NameFormat="urn:oasis:names:tc:
      SAML:2.0:attrname-format:uri">
    <saml2:AttributeValue xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:type="eidas-natural:CurrentGivenNameType">Max</saml2:AttributeValue>
  </saml2:Attribute>
  <saml2:Attribute FriendlyName="DateOfBirth" Name="http://eidas.europa.eu/
    attributes/naturalperson/DateOfBirth" NameFormat="urn:oasis:names:tc:SAML
      :2.0:attrname-format:uri">
    <saml2:AttributeValue xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:type="eidas-natural:DateOfBirthType">1940-01-01</saml2:
        AttributeValue>
  </saml2:Attribute>
  <saml2:Attribute FriendlyName="PersonIdentifier" Name="http://eidas.europa.eu/
    attributes/naturalperson/PersonIdentifier" NameFormat="urn:oasis:names:tc:
      SAML:2.0:attrname-format:uri">
    <saml2:AttributeValue xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:type="eidas-natural:PersonIdentifierType">AT/SO/
        MiACFWU9RnPcnoTZzCXDb1Wlgv4=</saml2:AttributeValue>
  </saml2:Attribute>
</saml2:AttributeStatement>
</saml2:Assertion>
```

Listing A.4: SAML-assertion from an eIDAS node

Appendix B.

Screenshots

This section contains screenshots of the demonstrator from chapter 6.

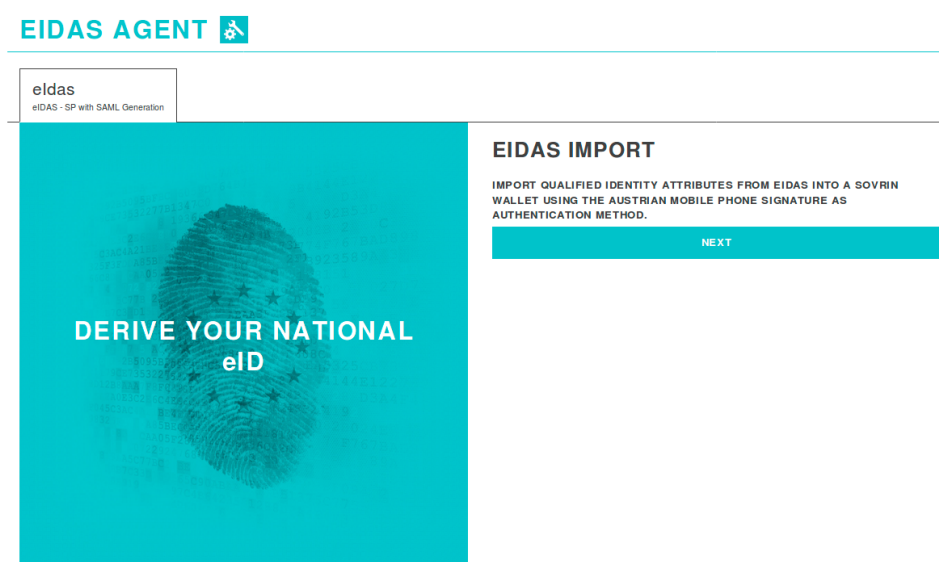


Figure B.1.: The demonstrator's starting page

Appendix B. Screenshots

EIDAS AGENT

DEMO-SP-CA
SAMLREQUEST GENERATED BY THE SP

DERIVE YOUR NATIONAL
eID

COUNTRY

RELAYSTATE

SAMLREQUEST

PHNhbWwycDpBdXRobUjlcXVlc3QgeG1sbnM6c2FtbDwPSJ1cm46b2FzaXM6bmFIZXM6dGM6U0FNTDoyLjA6cHJvdG9jb2wllHthbG5zOmcRzPzPSJodHRwOi8vd3d3LnczLm9yZy8yMDAwLzAsL3htbGRzaWcjlIB4bWxuczplaWRhcz0iaHR0cDovL2VpZGFZLmV1cm9wYS5ldS9zYW1sLWV4dGVuc2lvbnMlIHhtbG5zOnNhbWwycDpBdXRobUjlcXVlc3QgeG1sbnM6c2FtbDwPSJ1cm46b2FzaXM6bmFIZXM6dGM6U0FNTDoyLjA6YXNzZXJ0aW9uIiBDb25zZW50PSJ1cm46b2FzaXM6bmFIZXM6dGM6U0FNTDoyLjA6Y29uc2VudDp1bnNwZWNPZmIIZCgRGVzdGluYXRpb249Imh0dHBzOi8vdmlkcz05nd5hdC9lSURBU19ub2RIL2VpZGFZL0NvbGxiYWd1ZVJlcXVlc3QilEZvcmlQXV0aG49InRydWUllIEPSJlSURJbG9kaW50eS9lME777-1fGm74-AMkV9DhJlBQMM-BOV1TAM71VllLz-6E

HTTP POST BINDING
 HTTP REDIRECT BINDING

SUBMIT

RE-SIGN AND ENCODE

DECODE

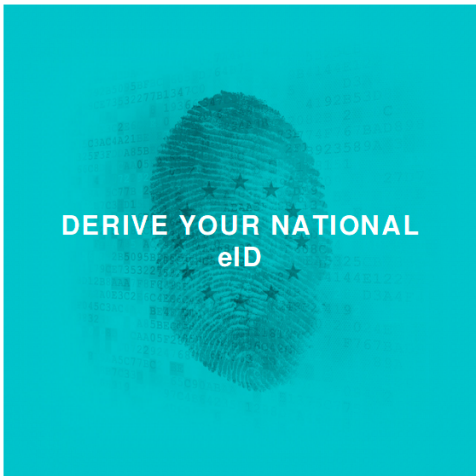
XML

<saml2p:AuthnRequest xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol" xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:eidas="http://eidas.europa.eu/saml-extensions" xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion" Consent="urn:oasis:names:tc:SAML:2.0:consent:unspecified" Destination="https://vidp.gov.at/eIDAS_node/eidas/ColleagueRequest" ForceAuthn="true"

Figure B.2.: SAML request creation

Appendix B. Screenshots

DEMO-SP-CA
SAMLRESPONSE RECEIVED BY THE SP



SAMLRESPONSE

```
PHNhbWwycDpSZXNwb25zZSB4bWxuczpzYW1sMnA9InVybipvYXNpczpuYW1lczp0YzptQU1MOjluMDpwcm90b2NvbCigeG1sbnM6ZHM9Imh0dHA6Ly93d3cudzMub3JnLzlwMDAvMDkveG1sZHNpZyMlIHhtbG5zOmVpZGFzPSJodHRwOi8vZWlkei51bnM6c2FibDI9InVybipvYXNpczpuYW1lczp0YzptQU1MOjluMDphc3NlcnRpb24iIENvbnNlbm909InVybipvYXNpczpuYW1lczp0YzptQU1MOjluMDpjb25zW50Om9idGFpbmVkaW50IIEEZXBzJWJWQGU0TDIzN0gtTGxRbmdlNUNQVWdDd1JaUWw1UUd4VksVm9uO
```

SUBMIT

RELAYSTATE

MyRelayState

ENCRYPTED RESPONSE

```
<saml2p:Response xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol" xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:eidas="http://eidas.europa.eu/attributes/naturalperson" xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion" Consent="urn:oasis:names:tc:SAML:2.0:consent:obtained" Destination="http://localhost:8080/SP/ReturnPage" ID="_Xjd.L9s7H-LiQngb5CPUWSwRZQI5OGxVORVon8QV_-EMlwa4FODEAZW485jxWb-" InResponseTo="_H5SMI20fLGS4x37xN.Gakq0USXvkPAOnbPyKDBIBISIMYHLn1ugmtyhCTMd1kTb" IssueInstant="2018-11-20T13:03:44.408Z" Version="2.0"><saml2:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity" xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion" https://vidp.gv.at
```

DECRYPTED XML ASSERTION

```
<saml2:Assertion xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion" xmlns:eidas-natural="http://eidas.europa.eu/attributes/naturalperson" ID="_1slqgmw1qDORwHIOEwN2cAu56CNj8yKabRB42jhAX9buE06xvFxsu5krhbz2MjS" IssueInstant="2018-11-20T13:03:44.408Z" Version="2.0"><saml2:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">https://vidp.gv.at/eIDAS_node/eidas/metadata</saml2:Issuer><ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#"><ds:SignedInfo><ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-
```

Figure B.3.: The response from the eIDAS node

Bibliography

- [1] Council of the European Union, *More secure transactions on the Internet*. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=LEGISSUM:310603%7B%5C_%7D1%7B%5C&%7Dfrom=EN (visited on 11/29/2018) (cit. on p. 1).
- [2] A. Tobin, D. Reed, P. J. Windley, and Sovrin Foundation, "The Inevitable Rise of Self-Sovereign Identity A white paper from the Sovrin Foundation," 2016 (cit. on pp. 1, 17, 18, 53, 68).
- [3] E. Bertino and K. Takahashi, *Identity Management Concepts, Technologies, and Systems*. 2011, p. 198, ISBN: 9781608070398 (cit. on p. 1).
- [4] L. J. Camp, "Digital Identity," 2004 (cit. on p. 1).
- [5] K. Cameron, R. Posch, and K. Rannenber, "Proposal for a Common Identity Framework : A User-Centric Identity Metasystem," *Identity*, pp. 1–30, 2008 (cit. on p. 2).
- [6] T. Ruff, *The Three Models of Digital Identity Relationships*. [Online]. Available: <https://medium.com/evernym/the-three-models-of-digital-identity-relationships-ca0727cb5186> (visited on 11/29/2018) (cit. on p. 2).
- [7] ISO/IEC JTC 1/SC 27, "ISO/IEC 15408-1:2009," *Information technology — security techniques — evaluation criteria for it security*, vol. 3, 2014 (cit. on pp. 3, 56).
- [8] Council of the European Union, "Regulation (EU) No 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC," *Official journal of the european union*, vol. 58, no. 910, p. 73, 2014 (cit. on p. 4).
- [9] —, "COMMISSION IMPLEMENTING REGULATION (EU) 2015/ 1501 - of 8 September 2015 - on the interoperability framework pursuant to Article 12(8) of Regulation (EU) No 910/ 2014 of the European Parliament and of the Council on elec," vol. 2010, no. 1316, 2015 (cit. on p. 4).
- [10] eIDAS Technical Subgroup, "eIDAS - Interoperability Architecture," no. November, 2015 (cit. on pp. 4, 6).

Bibliography

- [11] OASIS, "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0," no. September, pp. 1–93, 2005 (cit. on p. 6).
- [12] F. Hörandner and B. Zwattendorfer, "Secure Cloud Identity Wallet Assessment report on cryptographic technologies , protocols and mechanisms," 2017 (cit. on p. 7).
- [13] Österreichisches Bundesministerium für Digitalisierung und Wirtschaftsstandort, *Die Handy- Signatur*. [Online]. Available: <https://www.digitales.oesterreich.gv.at/documents/22124/30428/Handy-Signatur.pdf/006d35d9-2501-4162-b112-d60bc1eb1ab3> (visited on 11/29/2018) (cit. on p. 7).
- [14] A-sit, *KONFORMITÄTSBEWERTUNGSBESCHEINIGUNG ZUM KONFORMITÄTSBEWERTUNGSBERICHT I.S.D. ARTIKELS 20 DER VERORDNUNG (EU) NR. 910/2014 (EIDAS)*, 2014. [Online]. Available: https://www.a-trust.at/docs/Konformitaetsbewertungsbescheinigung%7B%5C_%7DA%7B%5C_%7DTrust.pdf (visited on 11/28/2018) (cit. on p. 7).
- [15] Digitales Österreich, *Aktivieren der Handy-Signatur*. [Online]. Available: <https://www.buergerkarte.at/aktivieren-handy.html> (visited on 11/29/2018) (cit. on p. 7).
- [16] W3C, *Decentralized Identifiers (DID) 1.0*, 2018. [Online]. Available: <https://w3c-ccg.github.io/did-spec/> (visited on 06/14/2018) (cit. on pp. 8, 9).
- [17] T. Berners-Lee, R. Fielding, and L. Masinter, *Uniform Resource Identifier (URI): Generic Syntax*, 2005. DOI: 10.17487/rfc3986. [Online]. Available: <https://www.rfc-editor.org/info/rfc3986> (visited on 06/14/2018) (cit. on p. 8).
- [18] M. Sporny and D. Longley, *Verifiable Claims Data Model and Representations*, 2017. [Online]. Available: <https://www.w3.org/TR/verifiable-claims-data-model/> (visited on 06/14/2018) (cit. on pp. 10, 11, 73).
- [19] Andrieu; Lee; Otto, *Verifiable Claims Use Cases*, 2017. [Online]. Available: <https://w3c.github.io/vc-use-cases/> (visited on 06/14/2018) (cit. on p. 11).
- [20] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *Acm transactions on programming languages and systems*, vol. 4, no. 3, pp. 382–401, 1982. DOI: 10.1145/357172.357176 (cit. on pp. 13, 14).
- [21] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," *Osdi*, 1999. DOI: 10.1145/571637.571640 (cit. on pp. 14, 16).
- [22] P. L. Aublin, S. B. Mokhtar, and V. Quema, "RBFT: Redundant byzantine fault tolerance," *Proceedings - international conference on distributed computing systems*, pp. 297–306, 2013. DOI: 10.1109/ICDCS.2013.53 (cit. on pp. 16, 17).

Bibliography

- [23] J. Law and L. Harchadani, *Scaling a BFT Consensus Protocol for Identity*, 2016. [Online]. Available: <https://github.com/WebOfTrustInfo/ID2020DesignWorkshop/blob/master/topics-and-advance-readings/scaling-a-bft-consensus-protocol-for-identity.md> (visited on 06/24/2018) (cit. on p. 17).
- [24] Sovrin Foundation, “Sovrin™ : A Protocol and Token for Self- Sovereign Identity and Decentralized Trust,” *Sovrin*, no. January, pp. 1–41, 2018 (cit. on pp. 17, 18).
- [25] D. Reed, J. Law, and D. Hardman, “The Technical Foundations of Sovrin A White Paper from the Sovrin Foundation,” no. September, 2016 (cit. on pp. 18, 19, 21).
- [26] Hyperledger Organization, *About Hyperledger*. [Online]. Available: <http://hyperledger.org/about> (visited on 07/13/2018) (cit. on p. 18).
- [27] J. Best, L. Boldrin, T. Brown, S. Conway, M. Chango, S. David, S. Fulling, N. George, D. O. Donnell, N. Hickman, A. Lake, J. Law, A. J. Poikola, M. Sabadello, P. Simpson, A. Tobin, and P. Windley, “Sovrin Provisional Trust Framework Sovrin Provisional Trust Framework,” no. March, pp. 1–27, 2017 (cit. on pp. 19, 30).
- [28] R. C. Merkle, “Secrecy, Authentication, and Public Key Systems,” no. May, 1979. DOI: 10.1.1.637.3952 (cit. on p. 21).
- [29] The Linux Foundation, *Hyperledger Indy Node*. [Online]. Available: <https://www.hyperledger.org/projects/hyperledger-indy> (visited on 07/19/2018) (cit. on p. 22).
- [30] —, *Hyperledger Indy Wiki*. [Online]. Available: <https://www.hyperledger.org/projects/hyperledger-indy> (visited on 07/19/2018) (cit. on p. 22).
- [31] Hyperledger Organization, *Indy Plenum Documentation*. [Online]. Available: <https://github.com/hyperledger/indy-plenum/tree/master/docs> (visited on 07/19/2018) (cit. on p. 22).
- [32] D. R. Morrison, “PATRICIA—Practical Algorithm To Retrieve Information Coded in Alphanumeric,” *Journal of the acm*, vol. 15, no. 4, pp. 514–534, 1968. DOI: 10.1145/321479.321481 (cit. on p. 23).
- [33] Ethereum Community, *Modified Merkle Patricia Trie Specification (also Merkle Patricia Tree)*. [Online]. Available: <https://github.com/ethereum/wiki/wiki/Patricia-Tree> (visited on 07/19/2018) (cit. on p. 23).
- [34] Hyperledger Organization, *Anoncreds Design*. [Online]. Available: <https://github.com/hyperledger/indy-sdk/blob/master/doc/design/002-anoncreds/README.md%7B%5C%7Danoncreds-workflow> (visited on 07/19/2018) (cit. on pp. 26, 27).

Bibliography

- [35] L. Ben, B. Dan, and S. Hanav, "Short Signature from the Weil Pairing," *Journal of cryptology*, vol. 17, no. 4, pp. 297–319, Sep. 2004. DOI: 10.1007/3-540-45682-1_30 (cit. on p. 28).
- [36] National Institute of Standards and Technolog, "Digital Signature Standard (DSS)," *Federal information processing standards publication (fips pub)*, vol. 186-4, no. July, 2013 (cit. on p. 28).
- [37] D. Boneh, B. Lynn, C. Gentry, and H. Shacham, *Aggregate and Verifiably Encrypted Signatures from Bilinear Maps*, 416-432. 2003, pp. 1–22, ISBN: 978-3-540-39200-2 (cit. on p. 28).
- [38] P. Dunphy and F. A. Petitcolas, "A first look at identity management schemes on the blockchain," *Ieee security and privacy*, vol. 16, no. 4, pp. 20–29, 2018. DOI: 10.1109/MSP.2018.3111247 (cit. on pp. 29, 30).
- [39] K. Cameron, *The Laws of Identity*, 2005. [Online]. Available: msdn.microsoft.com/en-us/library/ms996456.aspx (visited on 11/20/2018) (cit. on p. 29).
- [40] ShoCard, "Travel Identity of the Future," 2016 (cit. on p. 29).
- [41] ShoCard Inc., "ShoCard Whitepaper: Identity Management Verified Using the Blockchain," p. 20, 2017 (cit. on p. 29).
- [42] C. Lundkvist, R. Heck, J. Torstensson, Z. Mitton, and M. Sena, "Uport : a Platform for Self Sovereign Identity," 2016 (cit. on pp. 29, 30).
- [43] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash," *Journal for general philosophy of science*, vol. 39, no. 1, pp. 53–67, 2008 (cit. on p. 29).
- [44] G. Wood, *Ethereum: a secure decentralized generalized distributed ledger*. [Online]. Available: <http://gavwood.com/paper.pdf> (visited on 11/11/2018) (cit. on p. 30).
- [45] D. Gisolfi, *Self-sovereign identity: Our recent activity as a Sovrin Steward*. [Online]. Available: <https://www.ibm.com/blogs/blockchain/2018/05/self-sovereign-identity-our-recent-activity-as-a-sovrin-steward/> (visited on 11/20/2018) (cit. on p. 31).
- [46] NIST, "Guidelines for Derived Personal Identity Verification (PIV) Credentials," *Nist special publication*, pp. 800–157, 2014. DOI: 10.6028/NIST.SP.800-157 (cit. on p. 31).
- [47] F.-M. Kamm, "Definition of device system architecture and derivation scheme of mobile IDs," 2017 (cit. on p. 31).
- [48] Hyperledger Organization, *Indy Requests*. [Online]. Available: <https://github.com/hyperledger/indy-node/blob/master/docs/requests.md%7B%5C%7Dnym> (visited on 11/09/2018) (cit. on p. 39).

Bibliography

- [49] J. Camenisch and A. Lysyanskaya, "A Signature Scheme with Efficient Protocols," pp. 268–289, 2003. DOI: 10.1007/3-540-36413-7_20 (cit. on p. 42).
- [50] NIST, "Secure Hash Standard (SHS)," *Federal information processing standards publication (fips pub)*, vol. 180-4, 2015. DOI: 10.6028/NIST.FIPS.180-4 (cit. on p. 44).
- [51] D. Chaum, *Blind Signatures for Untraceable Payments*, 1983. DOI: 10.1007/978-1-4757-0602-4_18 (cit. on p. 68).