Universidade do Minho
Escola de Engenharia
Departamento de Informática

Bruno Miguel Gomes Fernandes

Self-Sovereign Identity
Decentralized Identifiers, Claims and Credentials using non Decentralized Ledger Technology

November 2021

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Bruno Miguel Gomes Fernandes

Self-Sovereign Identity
Decentralized Identifiers, Claims and Credentials using non Decentralized Ledger Technology

Master dissertation
Integrated Master's in Informatics Engineering

Dissertation supervised by
José Bacelar Almeida

November 2021

ACKNOWLEDGEMENTS

I would like to thanks everyone that supported me throughout the course of my academic years, especially during the witting of this master's thesis. I will start by thanking professor José Eduardo Pina Miranda for his help defining the thesis topic and for the weekly discussions that helped kick-start this research. I would also like to thank my supervisor professor José Bacelar Almeida for his guidance and feedback.

Finally a special thanks to my family, my girlfriend and my closest friends that accompanied and supported me throughout this journey.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

ABSTRACT

Current identity management systems rely on centralized databases to store user's personal data, which poses a great risks for data security, as these infrastructure create a critical point of failure for the whole system. Beside that service providers have to bear huge maintenance costs and comply with strict data protection regulations.

Self-sovereign identity (SSI) is a new identity management paradigm that tries to answer some of these problems by providing a decentralized user-centric identity management system that gives users full control of their personal data. Some of its underlying concepts include Decentralized Identifiers (DIDs), Verifiable Claims and Credentials. This approach does not rely on any central authority to enforce trust as it often uses Blockchain or other Decentralized Ledger Technologies (DLT) as the trust anchor of the system, although other decentralized network or databases could also be used for the same purpose.

This thesis focuses on finding alternative solutions to DLT, in the context of SSI. Despite being the most used solution some DLTs are known to lack scalability and performance, and since a global identity management system heavily relies on these two requirements it might not be the best solution to the problem.

This document provides an overview of the state of the art and main standards of SSI, and then focuses on a non-DLT approach to SSI, referencing non-DLT implementations and alternative decentralized infrastructures that can be used to replace DLTs in SSI. It highlights some of the limitations associated with using DLTs for identity management and presents a SSI framework based on decentralized names systems and networks. This framework couples all the main functionalities needed to create different SSI agents, which were showcased in a proof of concept application.

KEYWORDS Decentralized Identity, Self Sovereign Identity, DLTs, Decentralized Networks.

RESUMO

Actualmente os sistemas de gestão de identidade digital estão dependentes de bases de dados centralizadas para o armazenamento de dados pessoais dos seus utilizadores. Isto representa um elevado risco de segurança, uma vez que estas infra-estruturas representam um ponto crítico de falha para todo o sistema. Para além disso os *service providers* têm que suportam elevados custos de manutenção para armazenar toda esta informaçao e ainda são obrigados a cumprir as normas de protecção de dados existentes.

Self-sovereign identity (SSI) é um novo paradigma de identidade digital que tenta dar resposta a alguns destes problemas, criando um sistema focado no utilizador e totalmente descentralizado que oferece aos utilizadores total controlo sobre os seus dados pessoais. Alguns dos conceitos subjacentes incluem *Decentalized Identifiers* (DIDs), *Verifiable Credentials* e *Presentations*. Esta abordagem não depende de qualquer autoridade central para estabelecer confiança, dado que utiliza Blockchains ou outras *Decentralized Ledger Technilogies* (DLT) como âncora de confiança do sistema. No entanto outras redes ou bases de dados descentralizadas podem também ser utilizadas para alcançar o mesmo objectivo.

Esta tese concentra-se em encontrar soluções alternativas para a DLT no âmbito da SSI. Apesar de esta ser a solução mais utilizada, sabe-se que algumas DLTs carecem de escalabilidade e desempenho. Sendo que um sistema de identidade digital com abrangência global dependerá bastante destes dois requisitos, esta pode não ser a melhor solução.

Este documento fornece uma visão geral do estado da arte e principais *standards* da SSI, focando-se de seguida numa abordagem não DLT, que inclui uma breve referência a implementações não-DLT e tecnologias alternativas que poderão ser utilizadas para substituir as DLTs na SSI. Alem disso aborda algumas das principais limitações associadas ao uso de DLTs na gestão de identidades digitais e apresenta uma *framework* baseada em *name systems* e redes descentralizadas. Esta *framework* inclui as principais funcionalidades necessárias para implementar os diferentes agentes SSI, que foram demonstradas através de algumas aplicações *proof of concept*.

PALAVRAS-CHAVE Identidade Descentralizada, *Self Sovereign Identity*, DLTs, Redes Descentralizadas.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ABC Attribute-Based Credential

ABE Attribute-Based Encryption

ABS Attribute-Based Signature

CID Content Identifier

CKMS Cryptographic Key Management System

CRUD Create, Read, Update, and Delete

DAG Directed Acyclic Graph

DHT Distributed Hash Table

DID Decentralized Identifier

DIF Decentralized Identity Foundation

DKMS Decentralized Key Management System

DLT Decentralized Ledger Technology

DPKI Decentralized Public Key Infrastructure

GDPR General Data Protection Regulation

GNS GNU Name System

GUID Globally Unique Identifier

IPFS InterPlanetary File System

IPLD InterPlanetary Linked Data

IPNS InterPlanetary Name System

IRMA I Reveal My Attributes

JWM JSON Web Message

JWT JSON Web Token

KDF Key Derivation Function

LD Linked Data

MDAG Merkle Directed Acyclic Graph

OIDC OpenID Connect

PBKDF Password-Based Key Derivation Function

PKI Public Key Infrastructure

PoW Proof of Work

RTI Routing Table Insertion

SIOP Self-Issued OpenID Connect Provider

SSI Self-Sovereign Identity

SSO Single Sign On

URI Uniform Resource Identifier

VC Verifiable Credential

VP Verifiable Presentation

W3C World Wide Web Consortium

ZKP Zero-Knowledge Proofs

1

INTRODUCTION

1.1 CONTEXT

In today's world, our digital identities are scattered across multiple applications and services. Most of us have lost count to the number of registration forms we have filled, providing our personal identity information believing that identity provider can be trusted and will store our data securely. But this uncontrolled data proliferation and the service provider's inability to properly secure critical identity information have been proven to be the cause of multiple identity data breaches. Social Identity providers such as Facebook and Google, addressed this issue by centralizing identity, acting as the issuers and verifiers of user's identity. The current approach to digital identity is not secure, and its centralization poses a great threat to user's personal information. Besides that users have no control over what data is being collected from them by the identity providers.

The main reason for considering new models for identity is, among other things, to avoid this single point of failure in centralized systems and to put the user in control of his identity, allowing him to decide how much and what information to share with whom. In this new model of decentralised identity users are no longer dependent on any centralised service provider or trusted authority, having full control and ownership of their digital identity. The identity data is cryptographically verifiable, and stored locally on user devices creating a secure and portable digital identity. Putting the user in control of their identity prevents the misuse of personal data by service providers, and reduces large scale data breaches, as service provider no longer need to store client personal data, but just need to verify it's trustworthiness.

With Self Sovereign Identity (SSI) user's personal data is no longer controlled by service providers and identity can be validated by accepting a Verifiable Credentials issued to the user by a trusted authority. These credentials are cryptographically signed to ensure the trustworthiness and integrity of data contained on it, and it's ownership can also be cryptographically proven, due to the use of Decentralized Identifiers. This model of digital identity tries to resemble the one used in the physical world where trusted authorities issue credentials that citizens use to identify themselves or prove to have a skill or qualification.

Decentralized identifiers (DIDs)[1] also play a major role in the SSI ecosystem. They are unique identifier, usually created through cryptographic operations, that are associated with a DID Document. These documents contain information such as public keys, services endpoint and authentication data, and are often stored on decentralized networks. It is fully independent from any central authority as a user can create an identifier by generating a public/private key pair, and then storing the public key within the DID Document on the decentralized

network. DIDs represent the lower layer of the SSI Stack and provide the trust anchor needed to create a fully decentralized identity management system.

1.2 OBJECTIVES

The purpose of this study is to investigate the state of the art of SSI, as it is an emerging concept in the field of digital identity that promises to revolutionise the way people share and user their personal data online. The focus is putted in finding alternatives to DLT, commonly used in SSI systems as Verifiable Data Registry and source of trust. Given that DLTs do not provide the most secure nor scalable solution to the problem, the the main objective of this thesis is to evaluate the viability and challenges of implementing a SSI management system that is fully independent from DLT.

In a more detailed way, the objectives of this thesis are:

- Develop software components for the various participants of the SSI ecosystem, including Holder, Issuer, Verifier and Verifiable Data Registry. The registry must be developed using non-DLT as it is the main focus of this work. These components will also follow the W3C's standards for Credentials and Decentralized Identifiers.
- Research and analyse platforms/technologies that can be used as alternatives to DLT in the context of SSI, as well as evaluate already existing open source tools that can help implement the software components described above.
- Apply the developed software to real world use cases.

2

STATE OF THE ART

2.1 EVOLUTION OF DIGITAL IDENTITY

A digital identity consists in a set of attributes that describe an entity, it can be either a person or other things such as machines or IOT devices in a network. This set of attributes can then be used to uniquely identify the entity in a particular context.

Over the years digital identity has evolve through different stages and paradigms in order to solve the problem that is having a trusted and verifiable digital identity that users control. In [2] , Christopher Allen gives an overview of the four main stages in digital identity evolution, which are presented bellow.

2.1.1 *Centralised Identity*

In the first stage of digital identity most services used a centralized approach, where they had full control over users data. In this approach the digital identity is not tied to the user but rather to the service - if the service stops to exist the user identity disappears as well.

This approach works well within each service, but with the huge amount of services and applications in today's world, it forces the user to have a replicated identity for each service it uses, creating huge amounts of duplicated data. This approach is also very costly for the centralized authorities that have to store and secure user's data, as they has to comply with security standards such as the General Data Protection Regulation (GDPR)[3] or Personal Data Protection Act (PDPA)[4].

Although digital identity continued to evolve most of today services still use this centralized approach to identify their users.

2.1.2 *Federated Identity*

The next stage in the evolution of digital identity tries to answer the problem of data duplication by providing a way to authenticate users in multiple unrelated services with the same identity. Users no longer need to have a separated digital identity for each service they use, they can now have just one identity and use it to authenticate themselves in multiple services. Facebook and Google Login are a prime example of federated identity, as users can authenticate themselves in multiple third-party services using their Facebook or Google identity.

Although Federated identity reduces the problem of data duplication, it further centralizes identity management, by giving centralized authorities full control over user's identity and personal data.

2.1.3 *User-Centric Identity*

User-centric identity puts users in the center of the digital identity paradigm, giving them the ability to control and store their own data. Users have a personal data storage where they store their sensitive information, and then share it with third-parties as they please. It is important to note that the transaction of information is always started by request of the user and only occurs with its consent. The main focus of this approach is to empower users giving them a central role in the identity ecosystem allowing more control over their data.

The only problem is that users still have to rely on centralized identity providers, although they have more control over their data their identity is still owned by a centralized authority.

2.1.4 *Self-Sovereign Identity*

Self-Sovereign identity is built upon the the same concept of user-centric identity, the user is in the center of the ecosystem, and has the ability to control their personal data. However, in self-sovereign identity a user's identity is not tied to any centralized authority, meaning that the digital identity exists independently from any identity or service provider. Users have now full control over their identity and data, given the ability to decide what personal information they want to share and with whom they want to share it. It is also important to note that users have the right to withdraw consent to access their personal information at any time.

Centralization and the identity provider inability to securely manage user's data has been the root of many problems in previous approaches. SSI removes this factor from the equation providing a decentralized approach to digital identity.

These were the major paradigm shifts that happened to digital identity over the past few years. With centralized identity every system used his own centralized database to store user's information, however this approach resulted in user's having their data replicated and scattered around multiple web services. Some identity provider tried to find a solution to this problem providing users a way to authenticate in various services using the same digital identity. More user-centric approaches have followed that tried to empower users and give them control over their identity. SSI is one of these models and promises to be a promising concept in the field of digital identity. This thesis focus on this identity paradigm, highlighting it's privacy preserving mechanisms, and ability to ensure trustworthiness of user's identity and personal data in a fully decentralized manner.

2.2 SELF-SOVEREIGN IDENTITY

As it can be seen SSI is the result of years of evolution in identity management systems. Even though it is a fairly recent concept, it has the potential to change the way people identify and authenticate themselves in the digital world. This section provides a high level overview of SSI, starting with it's core concepts and properties. Then the architecture and the main participants of the ecosystem are also presented, to further explain how trust is enforced in a fully decentralized identity system.

2.2.1 *Definition and Concepts*

Self-sovereign identity (SSI) is an identity management paradigm in which identity owners have full control over their digital identity and personal data. With Self-sovereign identity entities store their data in encrypted wallets of their trust, and when needed, data is provided for verification. This approach does not rely on any centralized identity provider or repository, since users are identified though *Decentralized Identifiers* (DIDs), which are cryptographically verifiable and stored in a decentralized manner.

These identifier are the trust anchor of any SSI system, as they not only provide a way to uniquely identify each peer, but also have associated to them a set of cryptographic keys and endpoints used to authenticate and interact with the peer that controls DID . DIDs are then used on Verifiable Credentials (VCs) and Claims as a way to identify the subject of the credential. In SSI credentials provide a cryptographically verifiable and tamper-proof way of storing personal data or any other type of data associated with an entity. They are validated based on cryptographic methods, usually signatures, but also on trust established between the authority that issued the credential and the one verifying it. For example, a credential that is a digital representation of your passport or ID document must be issued by a governmental authority with legal power to do so. If not, the entity verifying the credential should reject it because the issuer is not trusted for issuance of that credential.

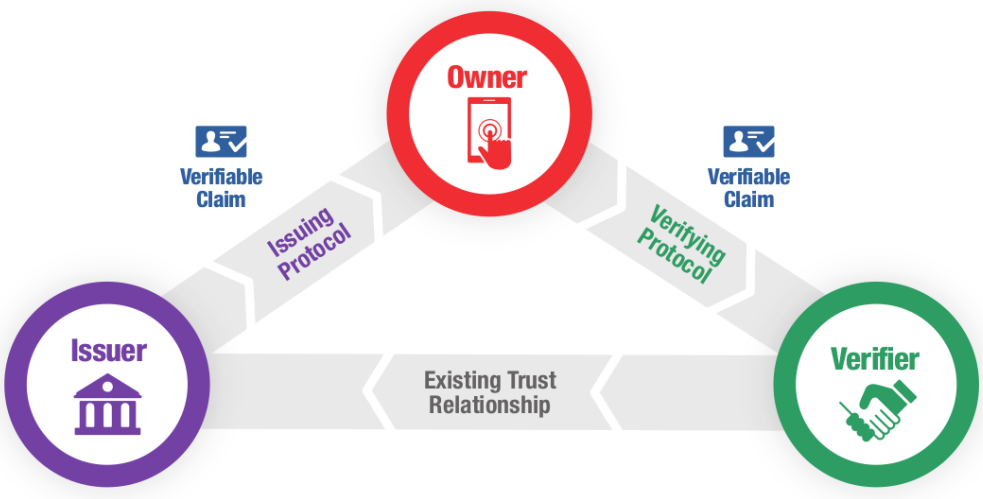


Figure 1: SSI participants [5]

SSI empower users by giving them full control of their digital identity, but with increase control also comes more responsibility. Users are now responsible for maintaining the trustworthiness and privacy of their identity and associated data. To prove the trustworthiness of their data users must present evidences of this information to a trusted authority, an Issuer. After verifying the correctness of this data, which sometimes can involve presenting real-word documents, the trusted authority issues a credential containing the data and an additional *Proof*. The *Proof* is usually a digital signature of the user's data, embedded in the credential and signed by the trusted authority. This way entities verifying the credential just need validate this proof to grantee the trustworthiness of the credential.

In short SSI offers a digital identity that is independent from any service provider and fully controlled by the users. By doing so it reduces the security risks and the cost associated with managing and storing data, as each user controls and stores its own identity in a decentralized manner.

To better understand the concepts involved in SSI, some of the terminology used throughout this document is described bellow.

Identity - In the context of SSI an identity is represented by a DID and the credentials associated with it, which can be used to identify an entity, whether a person, an organization, or a device. Each entity is not tied to one single identity as SSI allows entities to have multiple identities associated with them.

Claims - A claim is a statement made by an entity about itself or about another entity. Claims are usually a set of attributes that the entity is trying to prove. These claims must be cryptographically protected to assert their authenticity, and ensure they are not altered by malicious actors.

Credential - A credential is a set of one or more claims that are grouped in order to refer a particular aspect of an identity. Credentials can have various purposes depending on the claims they include, and like claims must always include a proof for authenticity and non-repudiation, which is often a digital signature.

Wallet - Wallets are used to store and manage credentials. There are two different approaches when it comes to storing and managing credentials, users can store their credentials locally on a device of their choosing like a phone or computer, or credentials can be stored and managed using cloud services.

DIDs - Decentralized Identifiers (DIDs) enable decentralized identity, by providing an identifier that is not dependent on any central authority. They are associated with an entity and often used in verifiable credentials. DIDs are independent from central authorities because they are based on public-key cryptography and stored on decentralized ledgers or networks.

DID Document - A DID is resolved into a DID Document that contain more information about the subject of the DID. This document contains information such as public keys, attributes about the subject, methods for

verification and service endpoints.

Decentralized Ledgers and Networks - Decentralized ledgers or networks are one of the fundamental building blocks of SSI, as they provide a decentralized root of trust for managing and registering DIDs. They also enable decentralized key management enforcing the control of the subject over their DIDs.

2.2.2 *Principles*

Self-sovereign identity has emerged from the desire of privacy and control over personal data. In [2] C. Allen defines the ten principles of self-sovereign identity which are a set of principles that an identity management system must fulfill in order to be considered self-sovereign. These principles presented by C. Allen follow a similar ideology of the seven laws of identity previously presented by K. Cameron in [6]. In more recent work [7] the authors propose one additional property(Provable) to the already existing ones proposed by C. Allen. An overview of these eleven principles is presented bellow.

- **Existence** - Users identity must exist independently from services or authorities. That is one of the main properties of SSI, the user is in the center of the ecosystem and his existence does not depend on any third-party.
- **Control** - Users must have absolute control over their identities. They have the responsibility to store it in their own wallets and the right to change it or hide it if intended.
- **Access** - Users must always have access to their personal data and claims that make up their identity. Data is accessed and updated without the need for any intermediary authority.
- **Transparency** - Algorithms used in self-sovereign systems should be free, well-known, open-source and independent from architecture. This enforces user's sovereignty and independence from any central authority.
- **Persistence** - Identities must be persistent and long-lived, although claims and personal data changes over time the identity itself is still the same. Persistent doesn't mean it's forever, users have the right to dispose their identity if they intend to.
- **Portability** - Identity and its data must be portable, in the sense that it should not be controlled or stored by any third-party, but either by the users, accompanying then everywhere they go.
- **Interoperability** - Someone's digital identity should be used across various services and applications. If identity was not interoperable between services, decentralization could never be achieved.
- **Consent** - The sharing of identity data and claims can only occur with the consent of the user. Even when claims about a user are presented by third-parties it must have the consent of the user to be a valid claim.

- **Minimalization** - In the process of sharing data the disclosure of personal information should be minimal. Only the necessary information should be disclosed in order to enhance user's privacy. Selective disclosure of user's personal data can be achieved using zero-knowledge techniques.
- **Protection** - The rights of users should be prioritized over the needs of the identity management system. To ensure this protection, authentications must be done using open-source algorithms in a decentralized manner.
- **Provable** - In order to trust claims there must be a way to verify if they are valid. This can be done through the use of cryptography and digital signatures, ensuring the trust between identities in a fully decentralized ecosystem.

However for the authors of [8] these principles are not enough to ensure total privacy and trust in the self-sovereign ecosystem. They state that the principles do not take into account scenarios like counterfeited or spoofed identities, weak verification of identities and identity transactions. So they propose three more properties to the ones stated above.

- **Usability** - The complexity of the identity system should be hidden from the users. Although the operations and mechanisms are a bit complex, user interfaces should be simple so the average user can control their identity intuitively.
- **Counterfeit prevention** - A malicious user should not be able to counterfeit a digital identity or acquire the identity of others. There must be mechanisms to ensure tamper-resistance identities and prevent channel sniffing attacks in order to mitigate the possibility of this scenario.
- **Secure transactions** - Users are able to share their personal data and identity with third-parties, however once that transaction is made it should not be possible for third-parties to read or tamper with that data.

These principles establish the foundations of a self-sovereign identity system. In these systems central authorities are no longer part of the ecosystem as user fully control and own their identities. Privacy is also a concern so users are given the ability to selectively disclose their personal data, making identity correlation harder to occur.

2.2.3 *Architecture*

In this section the roles and capabilities of each participants of the SSI ecosystem are presented. The image bellow summarized all the participants and its interactions, including the flow of credentials and the verification and registration of DIDs.

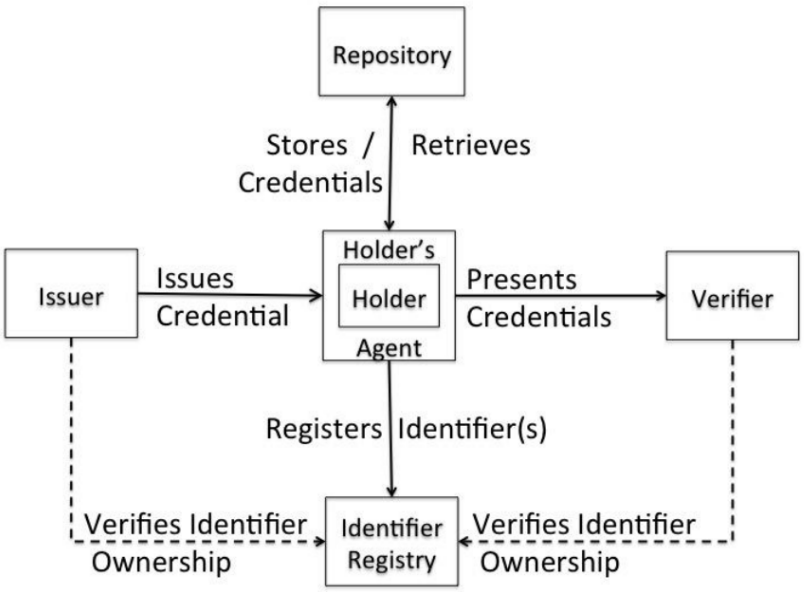


Figure 2: SSI Architecture [9]

- **Issuer** - The issuer is an entity who make claims about other entities. This claims are then put together into a verifiable credential, that is cryptographically signed by the issuer and transmitted to the holder. Issuer must verify the subject identifier in the identity registry before issuing the credential, to ensure its validity. Common issuer entities are corporations and governmental authorities like the ones who issue drivers licenses and passports.

- **Verifier** - A verifier is an entity that request credentials from holders, for validation purposes. The process of validating a verifiable credential includes verifying that the credential belongs to the holder and that he is in possession of the identifier presented in the credential. The trustworthiness of the issuer, status and proof of the credential are also verified, to ensure it has not been revoked by the issuer and it is still within its validity period. Examples of verifiers include service providers and web services that request credentials in order to provide access to resources or services.

- **Holder** - A holder receives credentials from issuers, stores them in a repository, and when needed credentials are provided to verifiers for validation. Holder also have the ability to create their own identifiers by registering them in the Identifier registry, so other entities can then verify ownership of this identifier in a decentralized manner. In most cases the holder of a credential is also the subject, however they can be different entities. For example a parent can be the holder of their child credentials who is the subject.

- **Repository** - Repositories are used to store credentials, this can be done locally on the holder's device or through cloud services. If credentials are stored outside the holder's device a protocol that ensures secure transactions and encryption of data must be used.

- **Identifier Registry** - The identifier registry is the entity responsible for managing and storing identifiers (DIDs) and DID documents. It manages cryptographic keys and metadata associated with each DID. The identifier registry is used as the source of trust to claim ownership of identifiers, and it should use decentralized technology to ensure the ecosystem's independence from any centralized repository. Example of verifiable identity registries include Blockchains, DLTs and Decentralized databases or networks.

3

SSI STANDARDS

3.1 DECENTRALIZED IDENTIFIERS

Decentralized identifiers (DIDs) are a type of globally unique identifier that is the building block of verifiable and decentralized digital identity. They are independent from any identity provider, so any participant in the SSI ecosystem can generate a new DID for himself at any time. These identifiers are generate in a way that its controller can cryptographically prove ownership over it. DIDs have URI like syntax and when resolved point to a DID document which contains cryptographic information, such as public keys, and other verification methods that allow authentication and interaction with the DID subject. In the DIDs ecosystem there are also DID Methods which specify the CRUD operations for DIDs and corresponding DID Document.

DIDs and DID documents are usually stored on DLTs, decentralized networks, or other decentralized systems, providing a root of trust without the need of a centralized authority. This is also due to the fact that DIDs are based on Public Key Cryptography, which means anyone can create a public/private key-pair without the need of a third-party.

The W3C is currently working on the standardization of DIDs [1] and has been actively developing work through the Credentials Community Group [10], specially on verifiable credentials and its underlying technology.

In their DID specification they start by defining the DID syntax. The most generic syntax is composed of a prefix string specifying that the identifier is a DID, known as a *DID Schema*, followed by the *Method Name* and *Method Identifier*, and each of the component is separated by a colon.

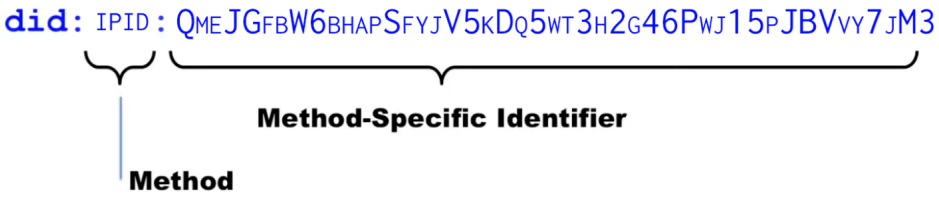


Figure 3: DID Syntax

It also defines the syntax of a *DID-URL* which is similar to the DID syntax but takes some additional arguments. The additional arguments are divided into *Path*, *Query* and *Fragment*. Besides that some method specific parameters can also be used before the *Path*.

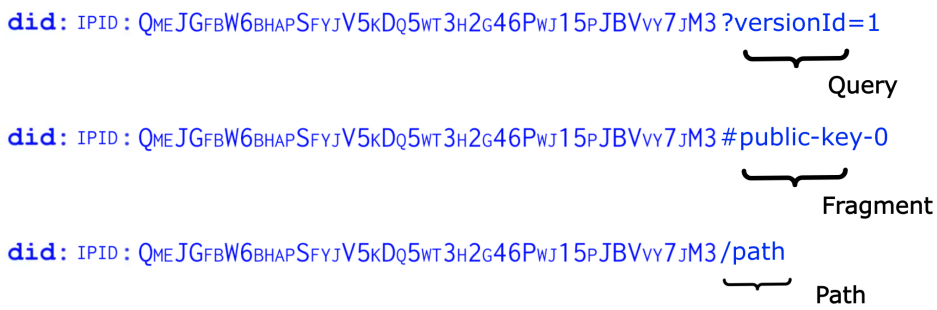


Figure 4: DID URL Syntax

The DID path and query are identical to a URI path or query, and are used to address resource through service endpoints, the fragment is used to identify or refer to some component inside a DID Document.

In the W3C specification it is stated that each entity of the SSI ecosystem should not have only one DID, but instead have one DID for each relationship it establishes with other entities. This is done to avoid correlation, as the DID is not only used for authentication, but also provides an encrypted channel between the two entities, allowing them to exchange data such as verifiable credentials.

To sum up the concept of a DID, it is a persistent identifier that can not be re-assigned, meaning that it always refers to the same entity even after being deactivated. Like URI it is resolvable and point to DID Documents that contain metadata such as cryptography keys that can then be used by the holder to prove ownership over the DID. It is fully decentralized and independent from any central authority or repository.

3.1.1 *DID Document*

The DID ecosystem can be viewed as a virtual key-value data store, in which DIDs are the keys and DID Documents are the associated values. These documents contain metadata such as public keys, authentication protocols and service endpoints, necessary to initiate a trusted interaction with the entity identified by the DID. DID Documents are represented using JSON-LD and are composed of a set elements that are described bellow.

```
{
  "@context": "https://w3id.org/did/v1",
  "id": "did:sov:123456789abcdefghij",
  "publicKey": [{
    "id": "did:sov:123456789abcdefghij#keys-1",
    "type": "RsaVerificationKey2018",
    "controller": "did:sov:123456789abcdefghij",
    "publicKeyPem": "-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n"
  }],
  "authentication": [{
    "type": "RsaSignatureAuthentication2018",
    "publicKey": "did:sov:123456789abcdefghi#keys-1"
  }],
  "service": [{
    "id": "did:sov:123456789abcdefghij;exam_svc",
    "type": "ExampleService",
    "serviceEndpoint": "https://example.com/endpoint/8377464"
  }],
  "created": "2018-02-08T16:03:00Z",
  "proof": {
    "type": "LinkedDataSignature2015",
    "created": "2018-02-08T16:02:20Z",
    "creator": "did:sov:8uQhQMGzWxR8vw5P3UWH1ja#keys-1",
    "signatureValue": "QNB13Y7Q9...1tzjn4w=="
  }
}
```

Figure 5: Example of a DID Document

- **Context** - The Context is used to establish terminology so that different entities processing the DID Document use a common vocabulary. It simply maps the terms used throughout the DID Document to the corresponding URIs.
- **DID** - The DID that resolves to the DID Document is also part of the DID Document itself. It identifies the entity the DID Document refers to, and is represented using the *id* property.
- **Public Keys** - A set of cryptographic keys that can be used for authentication purposes, or for establishing a secure communicating channel with the DID subject. They are also used for authorization of CRUD operations over the DID and DID Document.
- **Authentication** - Authentication methods are used to cryptographically prove ownership over a DID. In a DID document the authentication term is composed of an array of verification methods. These verification methods are normally public keys, but other methods such as biometric can also be used.
- **Service Endpoints** - Describe a set of endpoints for interacting and securely communicate with the DID subject. Each endpoint must have a unique *id* property, that should not be repeated within the array of endpoint, and the *type* and *serviceEndpoint* property must also be provided in order to identify the service type and URI.
- **Timestamp** - Timestamps are collected when a DID Document is created or updated, for auditing purposes. They are identified in the DID Document by the terms *created* and *updated*.
- **Signature** - Digital Signatures are used to provide integrity check for DID Documents, generally Linked Data signatures or JSON Web Signatures are used.

3.1.2 *DID Methods*

The DID Method is specified in the second component of a DID syntax as shown in section 3.1. It is a unique string that identifies the underlying system or network that supports the storage and management of DIDs and DID Documents. Each method must have a *method specifications* where the processes of creating, resolving and managing a DID on that specific network are defined. The method specification should also define how the *Method specific identifier* of the DID syntax is generated, as it must be globally unique and created without the need of any central authority. The other main purpose of a method specification is to define how CRUD operations are performed by the DID holder, and how proof of control and verification of a resolution responses is achieved. These operations can also be used to perform the common CKMS (Cryptographic Key Management System) operations, such as key registration, replacement, recovery and expiration.

Figure 6: DID Methods[11]

3.1.3 *DID Resolution*

DID Resolution is the process of obtaining the DID Document associated with a specific DID. Different methods have different ways to perform resolution, since they rely on different storage system to manage the DIDs. In a more generic term, resolution is performed using the READ operation defined by each method specification.

The W3C provides a DID Resolution specification [12] where the requirements and generic algorithms used in DID Resolution and DID URL Dereferencing are defined. This specification defines the input parameters passed to DID Resolver and the output format of the responses for both the resolution and dereferencing operations. Although the input parameters vary for the two operations the responses are the same, a DID Document or a *DID Resolution Result*, which contains parts or an entire DID Document and additional metadata. In the case of DID URL Dereferencing the response can also contain other types of resources. As for the input parameters of DID Resolution users can specify the response's format, between the two format described above, the version of the DID Document to fetch and additional caching options. For DID URL dereferencing process, input parameters specify the response's format, service type and if redirects should be followed.

The process of resolving a DID is accomplished though the READ operation as stated above, however there are two different operations that can be done, a *Verifiable Read* or a *Unverifiable Read*. The first one ensures the integrity and correctness of the response, by using secure channels of communication or other method specific approaches to ensure integrity of data.

A DID Resolver should support the resolution of various methods to ensure compatibility with different Identifier Registries, one example of such a resolver is the Universal Resolver [13] developed by the Decentralized Identity Foundation which supports the resolution of multiple DID methods.

3.1.4 *Public and Pairwise DIDs*

DIDs can exist in two different contexts, public and private. A public DID is one that is publicly stored on the DID registry and can be resolved, it has an high degree of correlation, as any entity can resolve the DID and access the information stored on the DID Document. So information that the user wants to keep private should not be associated with a public DID. They are usually associated with issuers of the SSI ecosystem that have a trusted public identity registered on the DID registry, so verifiers can then verify proofs issued by them.

A private DID also known as Pairwise-Pseudonymous DID is a DID that is only shared between a small number of parties. It is not stored on the DID registry so it is not resolvable or discoverable outside it's context. These DIDs are used when users want to avoid correlation and share information that should only be access within that context.

Other main advantage of private DIDs is that they provide better scalability for the whole DID ecosystem. Since they are not stored or resolved through a DID registry it reduces the load putted on registry, allowing for better scalability.

3.2 DECENTRALIZED KEY MANAGEMENT SYSTEM

A Decentralized Key Management System (DKMS) is an identity wallets standard, that specifies recovery and management operations for private keys and DIDs. The main purpose of this system is to provide storage and decentralized management of private keys, as well as backup and recovery mechanisms to ensure the identity owner doesn't lose control of their identity in case their keys get lost or compromised. Key rotation and revocation, storage of verifiable credentials and secrets are also part of the functionalities provided by a DKMS.

A DKMS must meet all the requirements of a Decentralized Public Key Infrastructure [14] (DPKI), which uses a completely different approach to key management compared to the traditional Public Key Infrastructure (PKI). The DKPI relies on DIDs and their underlying decentralized technology as the source of trust for storing and maintaining public keys, instead of relying on centralized certificate authorities (CAs). Blockchain and other DLTs are the most common solutions for providing this oracle of trust, however any decentralized secure system could be used. This decentralized approach ensures that end users will have more control over how their keys are generated and managed, and eliminates the need of certificate authorities to maintain the integrity of the key management system. This is one of the main advantages of DKMS as it provides no single point of failure, that if attacked can compromise a large number of users.

In the DKMS design, the DID specification solves the first problem of a DPKI by providing a way of uniquely identifying entities and verify their public keys in real-time. The DKMS specification builds on top of that and

addresses the challenges of generating and managing private keys and secrets related to DIDs, in a interoperable way that can be used across multiple devices. Besides key management, the DKMS enables identity owners to store and receive digital credentials, provides selectively disclose mechanisms and private encrypted channels for the exchange of keys and verifiable credentials with other peers.

Having this standard for digital identity wallets ensures a better interoperability and portability of keys and personal data, since identity owners will not be restricted to a single wallet provider and could at any time switch its data to another DKMS compliant wallet provider.

3.2.1 *Requirements*

A DKMS must meet a set of requirements in order to provide full control over keys and personal data. Most of these requirements are similar to regular CKMS requirement, but some of them differ mostly due to the fact that there is no reliance on centralized authorities to perform the key management operations.

According to [15] the DKMS requirements are the following:

- **Decentralization** - The main difference between DKMS and the traditional CKMS approach is the fact that it does not rely on centralized authorities to dictate the rules of the system. DKMS ensure decentralization by relying on DIDs to uniquely identify each participant, and their underlying technology to provide the oracle of trust for the whole system.
- **Privacy and Pseudonymity** - It must provide privacy and security mechanism such as encrypted communication channels and selective disclosure of personal data. Besides that it should enables anonymous interactions between peers through the use of pair-wise/pseudonymous DIDs, which also avoids identity correlation.
- **Usability** - It should be highly intuitive so entities with no knowledge on cryptography and key management can still easily use the system. A DKMS automates and abstract identity owners from most complex key management operations, without compromising privacy and security. To better enforce usability it should also provide compatibility with a broad number of applications.
- **Multiple Trust Models** - Due to its decentralized design, a DKMS must be flexible and support multiple trust models and frameworks.
- **Delegation and Guardianship** - Delegation of key management must be provided. Some identity owners may want others to manage their keys for convenience or safety purposes. Besides that other scenario in which delegation can be used is when an identity owner is not capable of identifying himself on his own. For example a parent may have guardianship over their child identity.
- **Portability** - A DKMS should enable identity owners to access and manage their keys, DIDs and personal data across multiple devices and applications. This is a fundamental requirement of DKMS since a fully independent key management system should not be restricted to a single device or application.

- **Extensibility** - A DKMS must be able to extend its support to new cryptographic algorithms, distributed ledger technologies and other security and privacy innovations.
- **Simplicity** - Having the complex task of key management, a DKMS must enforce simplicity in order to provide identity owners with a simple and straightforward way to manage their DIDs and private keys.
- **Open System and Open Standard** - The DKMS must be an open system based on open standards, in the sense that it should not rely on proprietary technology and protocols to be deployed.

3.2.2 *Architecture*

The DKMS architecture can be divided into three main layers as shown in Figure 7, the first layer is defined by the DID specification, since some key operations such as rotation and revocation may differ depending on the DID Method. The next two layers are defined by the DKMS specification, the Cloud layer which provides encrypted peer-to-peer communications channels, and the Edge layer where private keys are generated and stored. This architecture ensure interoperability of data and enables identity owner to manage their keys and personal data in real time across multiple devices.

The Cloud Layer is made of an agent and wallets that perform most of the key management, data storage and encryption operations which cannot be performed by edge agents due to limitations in availability, computational power or storage. This layer has a very important role in the DKMS architecture as it provides backup and recovery mechanics, so identity owners don't lose control of their identities in case private keys are lost, and enables key interoperability across multiple devices.

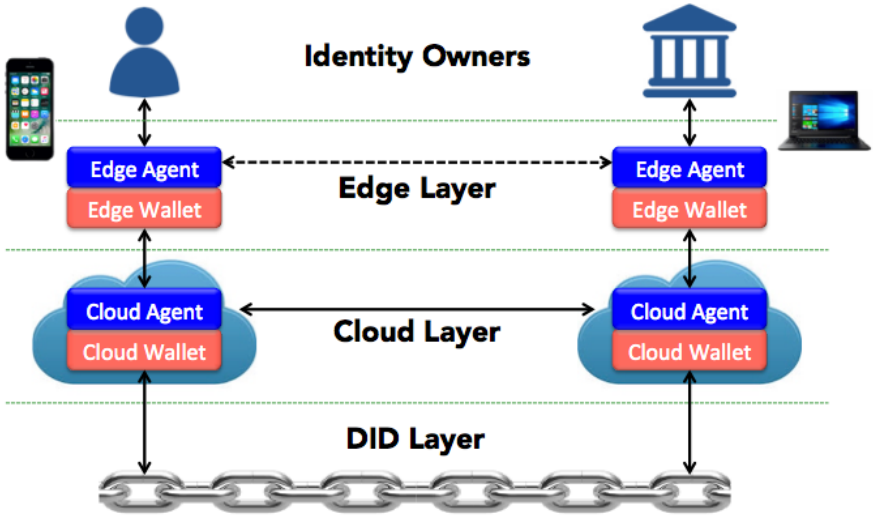


Figure 7: DKMS Architecture [16]

Besides that Cloud agents provide encrypted peer-to-peer communications channels for exchange and verification of DIDs, keys and Verifiable Credentials. Alongside with that it serves as the bridge between the Edge layer and the DID layer, providing DID resolution, access to DID Documents and performing operations such as key rotation and revocation on behalf of the identity owner. Although cloud agents are designed to be reliable and secure, they are the component that is most likely to be attacked, due to the fact that it is stored online, so security mechanism and protocols must be enforced to protect data handled by them.

The Edge Layer is also made of wallets, agents and mobile devices like smartphones or laptops that are under direct control of the identity owner and can be used to generate and securely store private keys and other secrets. Edge wallets are used for key storage because they are least prone to network intrusion than their cloud counterpart, although keys must also be backed up by the cloud layer in case the device gets compromised or lost, and control of the digital identity needs to be recovered.

Even though every edge agent is normally associated with a cloud agent to enable encrypted communication and key management operations, they can also be used to securely communicate with other agents using DID Communication, as well as to directly communicate with offline peers using protocols such as Bluetooth or NFC.

3.2.3 *Key Management*

A DKMS has to manage multiple key pairs in order to enable the various functionalities provided by the Edge and Cloud agents, and each one of these keys is used to perform a different task in the key management system. One of these keys is the *Master Key*, which is generated when a new identity is created and is used to authenticate access to the DID and to generate proofs of delegation. This private key should be kept in a secure Vault, backed up on an offline storage device or shared among peers because if it is lost or compromised it means complete loss of control over the digital identity. Because control of this private key is so important it's use must be minimized, so *Sub Keys* are used to perform DKMS functionalities such as data encryption and communications, as well as for verification of DID ownership through DID Authentication. These keys are also associated with a DID since they are stored in the DID Document as part of the identifier's data.

However *Sub Keys* themselves can be lost or compromised. Even though recovering from *Sub Key* loss would be much easier than recovering from *Master Key* loss, since the *Master Key* could be used to access the DID data and revoke or rotate the *Sub Key*. So since key loss and compromise will eventually happen a DKMS must have mechanisms to recover from it in order to provide a persistent digital identity.

The next sections will provide an overview of the mechanisms used by a DKMS to provide recovery, revocation and rotation of keys, as well as secure storage.

Key Recovery

If an identity owner changes or loses his device it also loses access to all the keys needed to manage his digital identity. To recover from this scenario a DKMS provides key recovery mechanisms that can be used to recover keys in case of loss. According to the DPKI specification [14] key recovery is achieved through **Offline Recov-**

ery mechanisms, where the identity owner creates an encrypted backup of their keys and wallet data using a recovery key, although this key could be generated using a KDF. Offline storage is used to store this backups, for example having a paper copy of your DID Master Key in QR-Code format or stored in a secure hardware device can be the solution to regain control of the DIDs in case the original key gets lost. Sub keys and any other data contained in the identity wallet can also be backed up offline, although it is not as usual due to the fact that each DID can have dozens of sub keys associated. Also sub key compromise can be solved through other mechanism such as key rotation and revocation.

Besides Offline Recovery, identity owners can also use **Social Recovery** mechanisms in which the backup is stored online in the cloud agent and the recovery key is cryptographically splitted into multiple shares that are then distributed to trusted peers. These peers must securely store their shares and make it available to the identity owner when requested. The identity owner can then recombine some of these shares to reconstruct the recovery key and decrypt the backup in order to regain access to keys and other wallet data. Shamir Secret Sharing [17], Fuzzy Encryption [18] and mnemonic seed phrases are some of the cryptographic schemes that can be used to implement key recovery mechanisms.

Even though offline and social recovery are the most common method for recovering access to keys in a DKMS, other approaches such as using biometric data could also be implemented.

Key Revocation

Key recovery mechanisms allow identity owners to regain access to their keys and private data in case of private key loss, but it cannot help them in case the private key gets compromised. In this case revocation mechanisms must be used to enable identity owners to list their revoked keys and present them to third parties like verifiers and service providers so they have a way of checking which keys can be trusted and which ones have been revoked or compromised. Forgetting a compromised key or erasing it from the wallet will not prevent an attacker from using it to impersonate the identity owner, so compromised keys must always be revoked.

Revocation mechanisms may differ based on DID Method or SSI system implementation, since the DPKI specification [14] does not define any standard mechanism for key revocation. While some methods may use a simpler approach like storing a revocation list on the ledger so third parties can access and verify it, others use more complex approaches like cryptographic accumulators [19]. This latter one is a concept where one way membership functions are used to tell if an element is part of a set or not, without revealing the actual elements in the set. It starts by mapping all the input data to prime numbers and calculates their product in order to generate the initial accumulator value, then membership is checked by calculating if a given number is a factor of the accumulator value. Adding or removing elements to the accumulator is achieved by multiplying or dividing the accumulator's value by the value of the element. The accumulator's value must be stored in a revocation registry on a ledger or other decentralized network for third parties to access and verify.

One final aspect of the key revocation is that access control to revocation lists and registries must be enforced in order to prevent identities other than the identity owner from updating or adding information to these registries.

Key Rotation

Keys associated with a DID, specially sub keys, should be rotated periodically in order to avoid tampering of data and identity impersonation. Master keys could also be rotated, although not as frequently, so identity owners can maintain persistent control over a DID even in if the private key gets compromise. The key rotation mechanism allows identity owners to replace old or compromised keys with newly generated ones, this can happen due to keys being revoked or expired, or simply because stronger encryption schemas were developed since. After rotation occurs all verification and signing operation will be done with the newly generated key pair. To better secure this mechanism each key rotation operation must be signed by the original private key so control over this key is proven before rotation is performed. Besides that expiration dates or validity periods should also be enforced in order to force keys to be rotated periodically even if they haven't been compromised.

Key rotation can also be very helpful in a scenario of passive compromise where the private key gets compromised without the knowledge of the identity owner. In this case, periodical rotation will eventually change the key and the attacker will lose the ability to decipher communications or authenticate as the identity owner. Although one problem arises from this, if the attacker performs a valid rotation operation to a key pair under his control he could lock the identity owner out of some of his services as well as authenticate as him. In this case key revocation must be used to remove the compromised key.

Key Storage

Another important aspect of key management is how private keys and other personal data are securely stored. In a DKMS this is achieved through the use of Encrypted Data Vaults [20], that may come in the form of hardware security modules or third-party cryptographic services. Encrypted Data Vaults provide encrypted storage in a way that storage providers cannot access or analyze the data. This is achieved by storing the vault access keys or data encryption keys outside the control of the storage provider, usually in the identity owner's mobile device or cloud wallet Keychain. To further enforce security, authentication mechanisms such as a Message Authentication Code (MAC) could be used in the encryption process to provide integrity protection on the data, and an authenticated TLS channel could be used to manage access to the vault.

Besides providing secure storage Encrypted Data Vaults also minimizes key exposure by keeping the keys always locked inside the vault. To perform operations such as encryption and signing or verification, data is sent to the vault and the operations are performed inside the vault so the keys never need to be moved or accessed from external libraries. In case an identity owner wants to backup a key to an offline storage device, and the key needs to be copied from the vault, a key encryption key should be used to protect the original key before it is stored outside the vault.

3.3 DID AUTHENTICATION

Addressing the problems of digital authentication has always been one of the main goals of the SSI paradigm. Since current authentication mechanisms have proven not to be secure and reliable anymore, mostly due to the identity provider inability to securely store and protect user's data, SSI defines an authentication mechanism based on public key cryptography and DIDs that is fully independent from identity providers. It intends to be a secure, simple and interoperable authentication solution, that can be used to authenticate identity owners across multiple web services with Single sign-on capabilities and increased privacy and security on user's data. By relying on DIDs and cryptographic keys as the source of trust, it unables service providers from tracking or correlating user's actions and information, as they no longer control the user's identity.

3.3.1 *Definition*

According to the *Rebooting the Web of Trust* paper [21] on DID Authentication (DID Auth), this mechanism is defined as the:

> "... ceremony where an identity owner, with the help of various components such as web browsers, mobile devices, and other agents, proves to a relying party that they are in control of a DID. This means demonstrating control of the DID using the mechanism specified in the DID Document's "authentication" object. This could take place using a number of different data formats, protocols, and flows. DID Auth includes the ability to establish mutually authenticated communication channels and to authenticate to web sites and applications."

So the main purpose of DID Auth is proving that an identity owner is in control of a specific DID. And considering that DIDs and DID Documents are cryptographically linked, proving control over a private key corresponding to a public key contained in the DID Document, also means proving control over the identifier. This proof of control is usually achieved through a challenge response mechanism, where the relying party generates a challenge, that is cryptographically signed by the identity owner, using the DID private key to generate a response. The relying party can them authenticate the identity owner by resolving it's DID and verifying the signature included in the response with the public key contained in the authentication property of the resolved DID Document. Most DID Auth protocols use this mechanism to provide proof of control over a DID, although challenge and response formats often vary depending on the protocol. The challenge can also include proof of control of the relying party's DID, so both participants in the interaction can mutually authenticate.

Cryptographic signatures encoded in JWTs are the most common proof method used with the Challenge Response mechanism, nonetheless other proof methods such as Verifiable Claims and Credentials can also be requested in the challenge. This is another approach, stated in [22], that can be used to authenticate users in the SSI ecosystem. However using VCs for authentication is out of the scope of DID Auth, and should be used only when authentication requires knowledge of more properties about the holder, other that just proving control

over a DID.

The exchange of challenge and response messages between identity owner and relying party can be done using different transport protocols, depending on the device used to perform the authentication. If using a web based app:

- the challenge can be sent via HTTP to a service endpoint defined by the identity owner in it's DID Document, and the response sent via the same method to a callback URL defined in the challenge, or
- the challenge can be encoded as a QR Code and displayed on a web site so the user can authenticate from a mobile device.

If the identity owner and relying party are physically close, protocols such as WiFi, NFC or Bluetooth could also be used to perform this exchange.

DID Auth can also be used to create authenticated communication channel that serve both the purpose of mutually authenticating the peers and providing a secure and encrypted communication channel to further exchange data.

3.3.2 *Protocols*

Since SSI is a fairly recent topic no protocol has yet been standardized to prove control over a DID. However there are a number of existing authentication protocols that rely on public key cryptography to perform authentication. These protocols can be used as a building block to implement DID Auth, or simply adapted to support DIDs in it's normal workflow.

In this section an overview over several potential DID Auth protocols is provided, specifying the data formats and mechanisms used by each one of them to authenticate an identity owner in the SSI ecosystem.

Self-Issued OpenID Connect Provider

OpenID Connect [23] is a decentralized authentication protocol that enables users to authenticate in multiple web sites without the need of a separate identity for each one. Authentication is performed through a third-party OpenID Identity Provider that controls the user's identity and sends this information back to web sites who request it on his behalf. Large corporation such as Google and Facebook are the most common identity providers, which often poses a security risk for user's personal information, since these corporations often collect, store and use this information without the user's consent.

The OpenID Connect specification [24] also defines the concept of Self-Issued OpenID Provider (SIOP) where the end user stands as its own identity provider instead of relying on a third party provider. In this case authentication is performed using a cryptographic key pair, the public key represents the identity and the user must prove

that it is in control of the corresponding private key in order to authenticate. This approach to authentication is very similar to the one defined in DID Auth, where an identity owner must prove control of the private key corresponding to the public key contained in the DID Document. Taking that into consideration the existing SIOP functionalities could be used to implement a DID Auth protocol, as shown in [25]. The authors further state that using SIOP to implement DID Auth would bring significant advantages when it comes to adoption, since most web sites already use OpenID Connect and developers are also familiar with the protocol.

More work on this topic has been conducted by the Decentralized Identity Foundation (DIF), which released a specification for a SIOP Profile compatible with DID Auth [26]. This specification defines a challenge response authentication mechanism based on JWTs, that reuses the existing OIDC infrastructure as well as message formats and combines it with DIDs, to create a fully decentralized and more secure authentication protocol.

Biometric Open Protocol Standard

Biometrics have become a common authentication method in the past few years. Not having to remember different passwords, but instead relying on unique characteristics of and individual such as fingerprints, iris or face recognition, not only has the advantage of usability but it can potentially be more secure, since biometric data is harder to forge than a regular password. However biometric authentication is not perfect, iris and face recognition algorithms can be tricked by photo or video spoof attacks, and fingerprint recognition can also be exploited. Besides that there is always the problem of where and how to store such sensitive information.

There are multiple approaches to how biometric data can be used in SSI, particularly in DID Auth. According to the authors of [21] biometric data can be used as a mean to protect and manage access to secret information, such as crypto keys, that are then used to perform DID Auth. Or it can be used as part of the DID Auth process as a proof mechanism instead of cryptographic signatures. In that case a reference to the biometric data must be contained in the DID Document, but the data itself must never be stored there. Instead biometric data must be securely stored by the identity owner and when DID Auth is performed some of this data is exchanged between the identity owner and the relying party.

The third and final approach stated by the authors suggests reusing existing biometric protocols such as the *Biometric Open Protocol Standard* (BOPS)[27] to extend DID Auth, making it support biometric authentication in it's normal workflow. Some of the advantages of using protocols such as BOPS to perform biometric authentication with DID Auth include having to exchange little to no biometric data between the identity owner and relying party during the authentication process.

Some work has already been developed on the topic of biometric in SSI, mainly [28][29][30]. The first two follow the approach of using biometric data as a proof mechanism in the authentication process. In [28] the authors use the concept of Cancelable Biometrics and bloom filters to apply transformations to the original biometric data, in order to securely store it and protect it against replay attacks. Moreover it suggest that biometrics could be used for offline authentication or in scenarios where the identity owner has lost control over their devices or keys.

In [30] a different approach was chosen, where the authors propose a protocol to extend the existing BOPS, that allows for the exchange of biometric credentials. This protocol makes use of the BOPS distributed storage schema to securely store the biometric credential data into multiple encrypted shares. However the biometric shares are not stored and managed by a BOPS server, but instead by the identity owners. These shares are then referenced in the DID Document so they can be used for authentication purposes.

3.4 DID COMMUNICATION

In the SSI ecosystem, entities have to interact with each other, in order to perform common tasks such as, credential issuance and exchange, proof presentation or simply share a text message. The foundations of every interaction in SSI are defined by DID Communication (DID Comm), a secure and private communication protocol build on top of DIDs. It relies on public key cryptography, anchored by DIDs, to provide encryption and authentication to every single piece of data exchange between peers.

Due to the diversity of devices and transport methods used to exchange data in the SSI context, DID Comm follows a simplex, message based approach to communication, that can be implemented over any transport protocols. In DID Comm messages are encrypted using the receiver's public key, accessed via the DID Document, and authentication is added using the sender's private key. This way only the receiver has the ability to decrypt the messages, since it is the only one in possession of the private key, ensuring that any third party that may spoof the message cannot access it's content. It is JSON based, more specifically it make use of JWMs (JSON Web Messages), and is divided into two main layers, the **Content Layer**, referring to the JWMs, and the **Envelope Layer**, an encryption layer used on top of JWM to authenticate and securely transport messages across the network.

DID Comm, has not yet been standardized, however DIF and Hyperledger Aries [31] have been working on that goal for some time. Initially Hyperledger Aries released a group of RFCs [32, 33] detailing how agent-to-agent communication would be performed in SSI, further specifying data formats and how encryption is performed on the data. Later DIF create the DIDComm Working Group [34], and picked up on the work developed by Hyperledger Aries, having defined a formal specification for DID Comm [35] and a set of guidelines to help implementers[36].

The next sections provide a more detailed look over the different layers of DID Comm and the set of protocols defined in the specification.

3.4.1 *Messages*

The Content Layer of DID Comm defines the plain text representation of a DID Comm message. At this level messages are serialized as JWM, witch implies the use of an additional header in every message. This header must include a unique identifier, created by the sender, a message type used to define the context in which the message is being send, which also helps the receiver to understand and validate the content of the message. It must also contain a timestamp for creation and expiration and an indication of the sender and receiver of the message. These last two fields are often DIDs but in some scenarios they can also be a DID URL that points to a specific key inside a DID Document.

Apart from the standard header, messages can also include *decorators*. A *decorator* is an optional field or set of fields used to store metadata. This data is mostly used to add additional semantic content to the message.

For example it can contain information about the thread handling the communication, which can be used to bind responses to the original messages. It can also contain information to help debug and trace messages, or handle error detection.

3.4.2 *Envelopes*

The envelope layer provides an encryption wrapper for DID Comm plain text messages. Similarly to a physical envelope, it hides the content of a message from anyone other than it's receiver, allowing messages to be securely sent over the network independently of the security mechanisms used by the transport protocol.

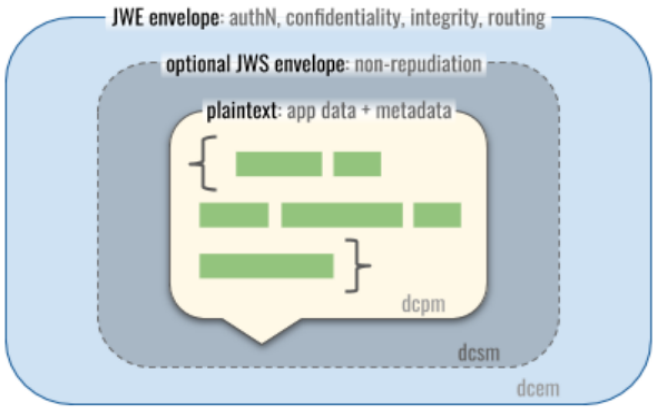


Figure 8: DID Comm Message structure [35]

DID Comm defines two different types of envelopes, which are used in different scenarios:

- **Encrypted Envelope** - This envelope is used when the receiver of the message is known beforehand. In this case the sender can resolve the receiver's DID and get the public key to encrypt the message. Encrypted envelopes support two different encryption formats, **Authenticated Encryption** which provides both encryption and authentication by adding sender's information to the encryption process, so the receiver can verify that the message was indeed encrypted by the specified sender. It also supports **Anonymous Encryption** where the sender information is omitted, and the receiver does not know who sent the message. However the DIF specification discourages the use of this encryption format, stating that instead holders should use pair-wise DID for interactions where they desire to maintain anonymity.
- **Signed Envelope** - This envelope is mostly used in scenarios where the receiver is unknown, so encryption cannot be performed. The Signed Envelope provides non-repudiation to DID Comm plain text messages through the use of digital signatures ensuring that messages are tamper resistant even when encryption is not used. It can also be used together with encryption, in this case messages must be signed before being encrypted and the signed envelope is used as the payload to create an encrypted one.

3.4.3 *Protocols*

The DID Comm specification defines a set of standard protocols that every agent must support. Each one of these protocols is identified by a message type, that defines the group of messages used in that particular protocol.

Routing

DID Comm messages may not always be delivered directly to the receiver, sometimes message have to pass between multiple intermediaries before getting to it's destination. These intermediaries are often cloud or edge agents working on behalf of identity owners. The DID Comm specification defines a *Routing* protocol that details how messages should be routed from the sender to it's final receiver, and how the different intermediaries should handle the incoming messages.

When routing is necessary to deliver a message to its receiver, the sender must attach this message to a new *forward message*. The original message is encrypted using the receiver's public keys, but is then attached to a *forward message* that is again encrypted, using a different intermediary's key. After receiving the message the intermediary decrypts the *forward message*, revealing the original message that is then delivered to the receiver. If more than one intermediary is used to route the message, this procedure is performed in each hop of the journey, creating a brand new forward message in every hop.

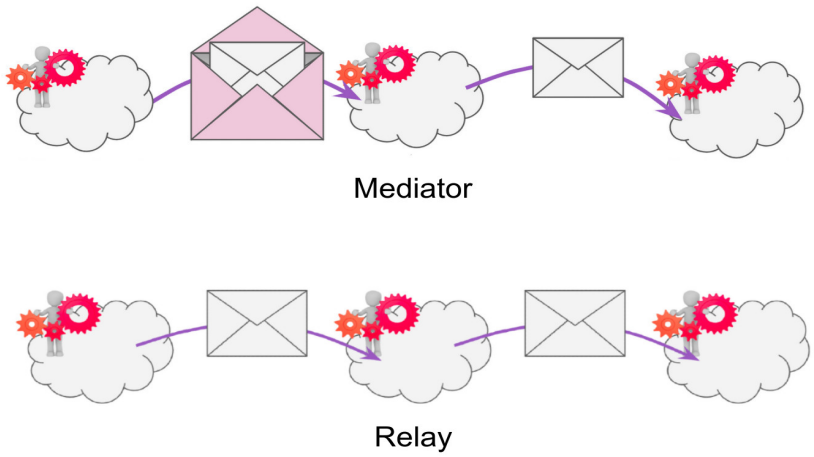


Figure 9: DID Comm Routing

The *Routing* protocol divides intermediaries into two different categories, *mediators* and *relays*. Mediators use the encryption procedure stated above, where they decrypt a *forward message* delivered to then, and create a new one for the next hop. While relays do not support encryption, they just forward messages as received, without wrapping it in a new *forward message*.

Feature Discovery

There is a multitude of different agent and devices interacting in the SSI ecosytem. Each one of these agents uses DID Comm to talk with one another, however every one of them has different features and can support different versions of a specific DID Comm protocol. There needs to be a way for agents to discover which features and protocol version another agent supports, so an agreement on these parameters can be achieved before initiating an interaction.

DID Comm defined the *Feature Discovery* protocol which specifies how agents can query each other to discover supported features. In this protocol agents are not obliged to reveal their features, instead feature disclosure is based on the level of trust associated with each relationship. An agent can revel his features to some agents but deny it to others. However agents must decide with care which entities they share their features with, since the feature discovery protocol could also be used as a way to fingerprint agents.

Trust Ping

Besides supporting different features and versions of DID Comm protocols, agents also differ when it comes to their availability. While cloud agents are always available and connected to the network, other agents, like most edge agent, cannot guarantee their connectivity at all times.

DID Comm defines the *Trust Ping* protocol, which provides a way to test availability and connectivity of an agent, while also ensuring the privacy and security of the connection. A successful trust ping request and response cycle not only confirms that the agent is available and reachable, but also guarantees that the encryption and privacy mechanism are set to initiate a secure communication.

3.5 VERIFIABLE CREDENTIALS

Verifiable credentials (VCs) are digital credentials that users control and store on their local devices, and can represent common physical credentials such as passports, drivers licence or college degrees. These credentials are used as a mean to identify an entity or assert properties and attributes about it.

In a more technical term, a verifiable credential is a set of claims asserted about an entity. These claims are tamper-proof and cryptographically tied (signature) to the issuer and subject of the credential, so that holders can then prove ownership over the credential and verifiers can validate it. Besides the subject and issuer it usually contains information such as the credential type, validity period, attributes about the subject, evidences or proofs and information related to terms of use.

The W3C is currently standardizing Verifiable Credentials and has been developing work through the Credentials Community Group [10]. This work already includes a Data Model [37] and Lifecycle [9] specification, as well as some guidelines for the implementation [38] and use cases [39].

The next subsections are going to provide a more detailed overview of verifiable credentials. First the concepts of *Claims*, *Credentials* and *Presentations*, and their role in the VCs ecosystem is going to be presented. Then an overview of the lifecycle is provided, detailing the operations involved in the issuance and verification of a VC, as well as other operations that can be performed with a credential. Lastly the Trust Model that enables this ecosystem to provide authentication and verification without needing any centralized authority is also presented.

3.5.1 *Claims*

Claims are statements made about an entity, used to assert attributes and properties that identify it's subject. They are one of the main component of a VC structure, since they contain the raw data that is expressed to the credential. Things like your address, age, college degree and even relationships you have with other entities can all be expressed through claims. The W3C specification [37] characterizes claims as a **subject-property-value** relationships that can be merged and organized in a graph representation.

These claims are usually signed by the subject and issuer of the claim to provide integrity and trustworthiness. This is not mandatory, as some claims can be unsigned, but it is a good practice to ensure better security.

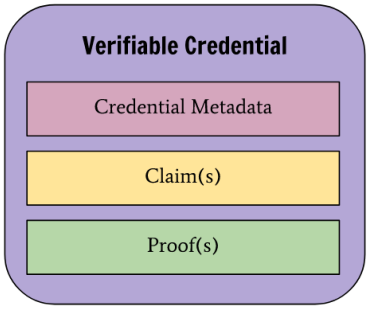
Claims can also be used in selective disclosure schemas to derive other statements about the subject in a privacy enhancing manner. The derived statement is then cryptographically signed to ensure it's trustworthiness. The most common example of this scenario is proving that a subject is within a certain age range without actually providing its date of birth. Support for selective disclosure in claims is important since it empowers user's consent and control over their personal data.

Sometimes additional evidence may be required in order to enforce trust in a claim, physical documents or biometrical data can be used as evidences so verifiers can assess the risk of relying in such a claim.

3.5.2 *Credentials*

A Verifiable Credential (VC) is a set of one or more claims, and other metadata such as public keys, for verification purposes, information about the issuer, type and validity period of the credential, etc. Besides claims and metadata credentials are also composed of *Proofs* that provide integrity and trustworthiness to the credential. In the SSI ecosystem only Issuers are allowed to create VC. To validate these credentials Holders must package them in a Verifiable Presentation, and share it with a Verifier.

The image bellow shows the abstract structure of a VC, as described above, and also provides an example expressed using JSON-LD, the standard syntax defined by W3C to represent VC.



(a) Abstract Structure

```
{
  "@context": [
    "https://w3id.org/credentials/v1",
    "https://example.com/travel-vocab/v1"
  ],
  "id": "urn:uuid:b306614c-73c7-11e8-b596-47e8c5ce9144",
  "type": ["VerifiableCredential", "PassportCredential"],
  /* gov't DID */
  "issuer": "did:example:CCnF3zFaXkPN4zB94XaomRdvw2zX3XHPVX3aExcgo6PV",
  "expires": "2020-01-01T00:00:00Z",
  "claim": {
    "id": "did:example:8vFBbPrhBUyG6DEzVncBZpzBNsmRrbfsQKXQKPLskBCu",
    "givenName": "Anand",
    "familyName": "Hamal"
    "citizenship": "US",
    /* any other claims made by gov't */
  },
  "proof": {/* signature by gov't */}
}
```

(b) JSON-LD Syntax

Figure 10: Verifiable Credential

Credentials start by defining a context to establish a common vocabulary and use shortcut terms throughout the credential. Each credential has a unique identifier defined by the property *id*, This identifier can be used for correlation, so when privacy is a concern, it should be omitted. The credential type is also part of the metadata, and in the example above defines the credential as a *Passport Credential*. Other important pieces of metadata are the *DID* of the issuer and expiration date of the credential, which are used in the verification process.

Then comes the set of claims that assert certain attributes about the subject of the credential. Each claim has it's own particular format and object fields in order to describe the specific attribute, but it must always contain a reference to the subject of the claim. Claims can be expressed using the *claim* and *credentialSubject* properties.

Lastly the Proofs are the component that make credentials verifiable. They provide a way to check integrity of the credential and verify authorship. The W3C specification defines two types of Proofs, external proofs which are not contained in the credential, like JWT, and embedded proofs which are part of the credential, like Link-Data signatures. Other commonly used proof methods are JSON Web Signatures and JSON-LD Proofs, and when selective disclosure is desired Zero-Knowledge proofs can also be used.

Depending on the type and requirements of the credentials, it can have other additional properties such as:

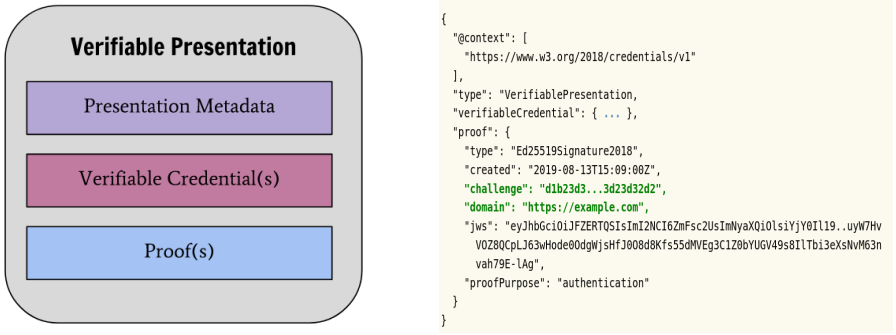
- *Data Schemas*, used to enforce a specific structure and validate syntax.
- *Evidences* such as physical documents used to establish trust on the claims being asserted.
- *Term of Use* which limit Verifiers's actions when validating the credential. This can include prohibiting credential storage or correlation or limiting the context or time-frame in which a credential is valid.

3.5.3 *Presentations*

A Verifiable Presentation (VP) is a set of credentials or data derived from credentials that is created by a holder in order to transmit or present information to a verifier. Like credentials, presentations also have a Proof component to ensure integrity and trustworthiness, and are encoded in a way that enables authorship to be verified. They usually group information from multiple VCs, enabling selective disclosure through derived credentials, that proves a property without it's value being explicitly contained in the presentation (Zero-Knowledge Proofs). If an

holder transmits a credential directly to an entity, it is transformed into a presentation. Credentials can not be transmitted on their own they need to be contained within a VP.

The image bellow presents the abstract structure of a verifiable presentation as well as an example using JSON-LD syntax.



(a) Abstract Structure (b) JSON-LD Syntax

Figure 11: Verifiable Presentation

A Presentation is composed of metadata such as the *context*, used to define common vocabulary, and *type* properties, which identifies the JSON-LD document as a Verifiable Presentation. An optional *id* property can also be used to uniquely identify the presentation. Credentials and other derived data are included in the presentation using the *verifiableCredential* property, which defines an array of objects, that must follow the VC syntax defined in the previous section.

The last component of a presentation is the *Proof*, which is similar to the Proof component of a credential. It provides integrity check and allows authorship to be verified. External and Embedded proofs are also supported in VP, and since selective disclosure of information is supported Zero-Knowledge proofs are also commonly used. This proof mechanism allows holders to prove possession of claims without having to revel the claim or credential itself. Usually a value is derived from a claim and asserted in a way that Verifiers can validate if they trust the issuer of the original claim. One example of such mechanism are Camenisch-Lysyanskaya signatures [40]. When credentials contained in a presentation use this type of proof mechanisms they must fulfill some additional requirements, such as having a *credentialSchema* property to refer the credential definition used to create the derived value. The presentation should also not leak any information that enables the verifier to correlate the holder.

3.5.4 *LifeCycle*

Now that the concepts of *Claims*, *Credentials* and *Presentations* are well defined, this section provides an overview of the lifecycle of a VC. It details the operations each entity is allowed to perform, and the data-flow of a credential in the SSI ecosystem. The focus will be on the lifecycle of credentials, although presentations usually follow a similar one.

The image bellow summarizes participants and it's interactions with a credential during it's lifecycle.

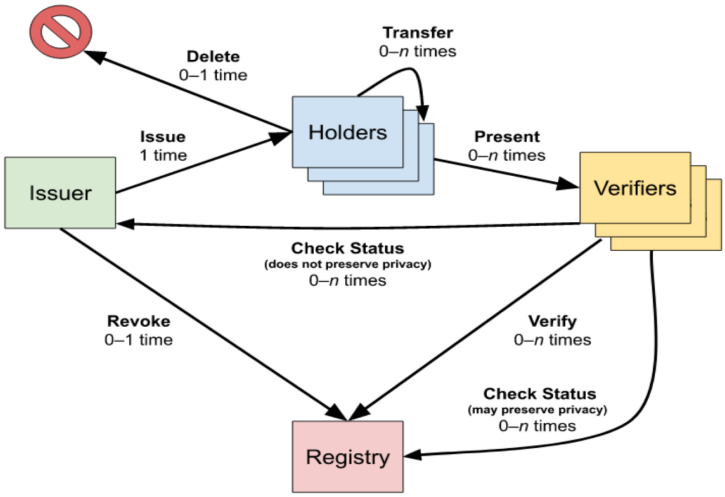


Figure 12: Verifiable Credential Lifecycle [37]

The lifecycle starts with a **Holder** requesting a credentials from an **Issuer**. To do so the Holder must have a DID registered on a public Registry, and may have to provide some evidences in order to create the claims. The Issuer has to validate this identifier, and check the trustworthiness of the evidence to ensure the Holder is entitled to the credential. Only then the credential can be issued and signed by the Issuer using the keys associated with his DID Document. Then the credential is counter-signed by the holder and stored in a wallet application, which stores credential locally on the user's device, thus enhancing control over personal data. This wallet application can provide additional security mechanisms such as encryption of credentials and mechanism to recover access to credentials in case of device being lost.

Issuers can also **revoke** credentials, so in the verification processes holders must provide the credential identifier to the verifier in order to check it's status. This verification process can include direct interaction with the Issuer or it is done by checking a revocation list stored on the registry. When privacy enhancing technologies such ZKP are used, the verification of credential status can be done without revealing it's identifier. Credentials can be revoked for two main reasons, firstly due to vulnerable cryptographic methods or compromised private keys, and secondly because credential data becomes outdated or invalid.

Holders use wallet applications to store and manage credentials, allowing them to visualize their credentials and track which personal information is being disclosed to each service providers. Besides that, holders can always ask for the revocation of a credential if they do not want to share that information with a service provider anymore, the credential is then deleted from the wallet. Holder should also be able to transfer credentials to other entities (wallets) or simply to a different wallet app of their choosing, it should not be necessary to request all the credentials again if a user decides to move to another wallet app.

Finally when a **Verifier** requests information, the holder must create a VP from the credentials contained in his wallet and present it, in order to obtain access to a service or resource. The process of creating a VP may

involve deriving data and other privacy enhancing techniques to allow minimal disclosure of information.

The **Verifier** then validates the authenticity of the data contained in the presentation by verifying the signatures of the Holder and Issuer. To do this it must accesses the Registry and obtain the public keys contained in the DID documents of both the Holder and Issuer. Before signature validation it must also ensure that the credential being presented is within it's validity period, and is entitled to the Holder - this is done by accessing the Registry and checking if Holder owns the DID contained in the credential. Besides this validation there must also exist a *trust* relationship between the Issuer and the Verifier. If this relationship does not exist the credential always fails the validation process - the next section will further discuss the Trust Model of the SSI ecosystem. Credential status must also be verified to ensure that it is not revoked, as stated before this process can be done by direct interaction with the Issuer or through the Registry.

This is the lifecycle of a VC in the SSI ecosystem. As it can be seen the Holder is a central piece in the lifecycle since most of the interactions start with his actions, like requesting and sharing credentials. This enforces user's consent and control over their personal data, and ensurers transparency in the process of managing and sharing digital identities.

3.5.5 *Trust Model*

This section will provide an overview of the Trust Model used in SSI. This model ensures the proper functioning of the identity management system without the need of a trust relationship between every participant. However this relationship must exist between some of the participants in order to successfully validate credentials.

To start, a Verifier must trust the Issuer in order to accept credentials issued by him. This trust relationship can be achieved by using a proof component in the VC specifying that the Issuer has indeed generated that credential, or if this component is not present, the credential must be transmitted in a way that ensures that the Issuer has generated the credential, and that it has not been tampered with during it's transmission or storage. The Issuer must also be a valid entity to issue the credentials, for example to issue a passport credential, the issuer must be a governmental entity with legal power to do so. This trust relationship is unidirectional since the Issuer does not need to trust or know the Verifier in order to validate a credential.

Besides that, all participants trust the Registry to be a correct representation of identifiers and it's associated data. The registry must be tamper-proof, to ensure trust and allow ownership and control to be proved in a decentralized way, where no central authority exists.

The Holder must also trust the wallet to securely store his credentials and not tamper or lose them while stored. If a Holder loses access to his Repository, there must be a mechanism to restore access, so user's do not have to request all the credentials again. The Repository should only release the credentials to it's Holder and no other entity.

This model differentiates itself from others by using a decentralized Registry as a source of trust. The Registry stores all the identifiers and information associated with it, being the trust anchor and building block of the hole SSI ecosystem.

4

A NON-DLT APPROACH TO SSI

SSI and Distributed Ledger Technology (DLT) [41] are two concepts that are often interconnected. In fact most SSI systems make use of DLT as a way to implement a DPKI, since it provides a decentralized, trusted and verifiable data registry suitable for management of DIDs and DID Documents.

DLT is a broad concept in itself which includes many different types of ledgers, but in an overall view it can be described as a distributed network of nodes that use a consensus protocol in order to handle updates to an append-only ledger. The ledger is replicated across every node of the network, meaning that every node maintains the exact same copy of the ledger, and when updates are needed the state of the ledger is agreed through the consensus protocol. Besides that, DLTs are immutable by design, so after data is written to the ledger it will no longer possible to delete or change it.

Most DLT approaches also implement an additional security layer through the use of public key cryptography. Where each node manages a asymmetric key pair that is used both to identify the node, through the hash of the public key, and to sign data recorded to the ledger, using the private key. Such security mechanisms combined with the ability to achieve consensus over the state of the ledger without the need for any central authority makes DLT a suitable candidate for the implementation of SSI systems.

However there are also some concerns associated with the use of DLT in the context of digital identity. Some of these concerns are related with intrinsic aspects of certain DLTs, such as the heavy replication of data, low performance, and the use of consensus protocol based on computational hard problems. These properties unable DLT systems to scale properly, which can be a major problem when trying to achieve a global identity management system, that could handle millions of records. Moreover there is the hurdle of data regulation policies, such as the GDPR, that specify a set of strict rules when it comes to handling personal and identity data. Some DLT approaches may not fully comply with these standards which is also part of the concern. These drawbacks created some scepticism on DLT being the best technology to anchor SSI, in fact the authors of [42] concluded their research saying that DLT is not the *silver bullet* for digital identity management.

This chapter will focus on a non-DLT approach to SSI, as it is the main focus of this thesis. First a brief reference to some research and related work on the topic of non-DLT SSI. Then an overview of some of the reasons and motivation behind choosing this approach are also provided, highlighting the major concerns over the DLT approach. Afterwards some alternative solution to DLT, that can be used in the context of SSI as

identity registries, are also provided, as well as some of it's limitations. And finally an overview over some implementations that follow a non-DLT approach.

4.1 RELATED WORK

To provide some background on the non-DLT approach to SSI, a brief overview over some of the research developed in the topic is provided bellow.

To start, [43] gives an comparison between some implementations of SSI systems, both DLT and non-DLT based, and also raises the question if DLT is required in order to achieve SSI. It uses the ten principle of SSI [2] defined by C. Allen, with the additional *Provable* principle defined by [7] as the evaluation criteria to compare the different implementations, see section 2.2.2. In the end although the DLT based systems meet more of the evaluation criteria, it concludes that DLT is not necessary to implement a SSI system.

Some of the non DLT implementation mentioned in this paper include the *Privacy Data System* (PDS) [44] which relies on a decentralized network, instead of DLT, to create an environment where users can store and share their private data in a self-sovereign way. In this network data is split into multiple encrypted chunks that are spread across the nodes. Unlike DLT, each node does not maintain a copy of the whole data, instead it stores only one of the chunk of each piece of data published on the network. Besides that each chunk of data is encrypted under a different partial key, and nodes do not know where the remaining chunks are stored and under which partial key, making it impossible for any single node to reconstruct the original data.

The other non-DLT solutions mentioned include, *reclaimID* [45], a SSI system constructed on Attribute-Based Encryption, that is built on top of the GNU Name System (GNS) [46]. ReclaimID as well as its underlying infrastructure are further detailed in [47], where the authors provide an in depth analysis highlighting its main features and properties. And finally *IRMA* [48], short for I Reveal My Attributes, is an identity management system based on the Idemix credential schema[49], which provides selective disclose of data as well as other privacy properties such as unlinkability. IRMA is also compared with *Sovrin* [50], one of the most relevant SSI solutions, in another research [51] which further claims that both solutions provide similar properties in terms of privacy and functionality. Both of these solutions will be described with more detail in a following section dedicated to non-DLT implementations.

More work on the topic includes [52] which suggest the use of intermediate certificates to implement a SSI solution. The proposed schema makes use of X.509 certificates combined with a graph data structure as a way to serialize and validate attestations about users. Besides describing their approach the authors also express some concern on the use of DLT to implement SSI solutions. They start by questioning the decentralized nature of current DLT implementations such as Sovrin and uPort [53], further stating that in addiction to the inefficiency and low performance associated with DLT, there are also some concerns when it comes to compliance with the GDPR. More specifically they refer the fact that DLT is characterized by immutable storage which can be a barrier for guaranteeing the right of erasure defined in the GDPR.

There is also more research that highlights the challenges and limitations of DLT in the context of digital identity management. In [54] the authors state that DLT is known for it's high latency, and low throughput of transactions. Given that SSI enables each user to have multiple identities, the transaction load on the network can be very high. So the underlying system must be able to support a heavy loads of transactions per second, which most DLTs cannot. Besides that, they also makes note that most known DLT solutions use the *Proof-of-Work* algorithm in order to achieve consensus on the ledger. This algorithm is very resource-intensive and consumes large amounts of energy, which is also viewed as a drawback associated with some DLTs.

On top of that some implementers like *Identiq* [55] equally consider that using Blockchain for identity management brings additional risk for data management and also raises the overall complexity of the system. Further stating that the use of a decentralized network based on multi-party computing and zero-knowledge cryptography would be a better approach.

4.2 WHY NON-DLT?

Although most SSI systems are anchored on DLT, there are a number of concerns and limitations when it comes to the use of this technology for digital identity management. As briefly mentioned in the previous sections, some limitations are associated with the lack of performance and scalability provided by DLT. This is a concern since it can become a bottleneck for identity management systems as they take adoption and grow to millions of users. In addition to that there are also a number of issues related to the decentralized nature of some DLT approaches as well as the compliance with the GDPR. Processing and storage of personal data is subjected to a set of rules and regulations, which some DLT's may not be able to fully comply with due to intrinsic features such as data replication and immutable storage.

This section provides a more in depth overview over some of the limitations of DLTs in the context of decentralized digital identity.

4.2.1 *Scalability and Throughput*

DLT is often associated with low performance and lack of scalability. This is mainly due to it's low throughput, or high latency when recording transactions to the ledger. In fact the most known instances of DLT, the Bitcoin and Ethereum blockchains are limited to 7 and 15 transactions per second [56, 57]. The low throughput and performance is usually related to the consensus protocol, often based on computational puzzles or other time costly mechanisms, as the task of registering a new transaction to the ledger must be computationally hard. Besides limiting the throughput and scalability some consensus protocol are associated with high energy consumption's which can also be a major drawback when considering DLTs as a solution for decentralized identity management [54].

So the low throughput of DLT networks greatly reduces their scalability, as they are always limited by the small rate of transactions it can handle. Furthermore this limitation is aggravated by the fact that SSI enables users to manage multiple identifiers or *personas* [54]. This can result in a a huge number of transaction being registered

in the network which can degrade the overall performance and quality of the service as the network becomes bigger and has to handle transactions for millions of users at a time.

This scalability problem may not apply to every single DLT, as it is a vast concept with many different approaches, types of ledgers and consensus protocols. Different ledgers can provide distinct performance and scalability properties, since these properties are essentially determined by the consensus protocol being used.

4.2.2 *Transaction Fees*

Besides scalability another major drawback associated with most DLTs are the transaction fees charged every time someone wants to write to the ledger. This is more common on public permissionless DLTs, although some permissioned approaches like Sovrin also charge additional fees for some of the SSI operations. These fees can range from small fees for creating and updating DIDs, like in the Verse One ledger where the DID creation and update costs 0.89$ and 0.25$ [58], to considerably high one like the case of Sovrin where DID creation costs 10$ while creating a credential schema or revocation registry costs 50$ and 20$. Sovrin estimates that an issuer that does not create credential schemas or revocation registries would spend an at least 35$ to start issuing credentials in the Sovrin ecosystem, and if it used revocation and schemas it would spend at least 105$ not considering the fees of updating the revocation registry which are 0.10$ for each update [59].

This can be a serious obstacle to adoption, in an environment where identity owner are advised to create multiple DIDs to differentiate personas and avoid correlation, having such high fees for these operations may discourage the use of these systems. Above all a SSI solution should not have associated costs or fees, at least not for the creation and management of DIDs. Using a permissioned DLT approach would solve the problem of transaction fees, however entities using this approach choose to charge fees that are sometimes bigger that permissionless DLT transaction fees.

4.2.3 *Decentralization*

To avoid some of the limitations mentioned above several SSI systems choose to rely on permissioned DLTs. While using this approach often results in better performance and scalability, it can put in question the decentralization of the whole system. Unlike public DLTs, where anyone can run a node and become part of the network, in the permissioned approach there is a central entity who controls and grants access to the ledger. This entity has absolute control over the ledger and decides who accesses it and who takes part in the ecosystem. By relying on a central entity to govern the ledger, the permissioned approach completely defeats the purpose of SSI and decentralized identity management.

Another thing that can also lead to the centralization of the ecosystem is the heavy data replication mechanism enforced by most DLTs. Every node in the network needs to store the entire ledger, and overtime this can create storage constraints for weaker nodes with less computation and storage power. As the ledger grows bigger and reaches hundreds of GBs, smaller nodes will be unable to store it and the network will gradually grow to a

network of less but more powerful nodes. This can affect the decentralization of the system and it will result in users delegating the storage and management of the ledger node to third parties, given the storage constraints.

4.2.4 *GDPR*

Some of the most significant concerns when it comes to using DLT for digital identity are related to compliance with data protection regulations such as the GDPR [3]. These regulations define a set of requirements and rules that must be enforced when handling and processing personal information, and a few of DLT's properties may be directly incompatible with some of the requirements defined by such regulations. In the context of SSI DIDs, DID Documents, and Credentials can be considered personal data as they provide a way to either identify or provide identifiable characteristics about a subject. Although there can be some debate on DIDs being considerate personal data, their use over a long period of time, for the issuance of multiple credentials can be used to link multiple attributes about a subject. Both authors of [60, 61] consider DIDs and DID Documents as personal data and thus under the scope of GDPR.

As DID Documents are often stored using DLT, they must comply with these regulations, which sometimes is not possible due to intrinsic characteristic of certain DLT solutions. One such characteristic is the replication of the ledger in every single node of the network. This characteristic stands as a direct violation of the Data Minimization principle defined in the GDPR, as the storage and handling of personal data must be reduce to the minimum needed [62]. Storing the exact same data in every node, although needed to maintain the integrity of the ledger, creates "unnecessary" copies which can further increases the risk of data being exposed. Besides that the append only nature of most DLTs is also a big obstacle to compliance, since it make it impossible to ensure the right to be forgotten. According to the GDPR users have the right to ask third parties to erase any personal data associated with them, and given that most DLTs only allow for data to be added, and never deleted, the right of erasure becomes almost impossible to ensure. [62, 60].

As it can be seen, DLT is far from being the perfect match for SSI. Its multiple limitations, not just in terms of performance and efficiency but also concerning compliance with data protection regulations, makes one wonder if this is the best infrastructure to support SSI. DLT is not the only decentralized computing infrastructure out there, in fact there are a few other decentralized technologies that can provide similar assurance levels while even tackling some of the limitations of DLT. Such decentralized technologies could provide better performance and scalability as they do not have to bare the overhead of a consensus protocol and the heavy replication of data.

4.3 ALTERNATIVES TO DLT

Most SSI systems use DLT, as a way to implement a DPKI, due to its decentralized nature and ability to achieve consensus without the need of a centralized authority. Still the decentralized trust in which SSI relies is anchored on the DPKI itself and not on any specific feature of DLT. This meaning that other decentralized infrastructures can potentially fulfill the requirements of SSI and be used as decentralized identity registries, for management of DIDs and DID Documents. Given that any decentralized network removes the need of a centralized authority, and considering that some of the features of DLT only create an additional burden, as shown in the previous section, this thesis focuses on alternative systems that could replace DLT in the context of SSI.

In this section some alternative decentralized infrastructures are presented, which can be used to support DID Methods for SSI. Most of them are decentralized name systems and networks based on Distributed Hash Tables (DHT) [63] and public key cryptography, which provide similar security properties to DLT. Although they are not limited in scalability and performance as they do not rely on a consensus protocol to maintain the integrity and validity of the data.

Also data is not replicated in every node of the network, instead it is usually split into multiple chunks that are scattered across the network. Unlike DLT, no single node contains the whole database, which further secures and protects the data. This slight difference in the storage mechanism of the network could ensure compliance with the Data Minimization principle defined by the GDPR, as data is no longer overly replicated. Besides that alternative solutions such as the IPFS (InterPlanetary File System), even if characterized by immutable storage, provide mechanisms to erase data by using a garbage collector which removes files that has been unpinned from the nodes. This may also be an answer for compliance with the GDPR right to be forgotten, as data can be erased from these networks. With that in mind bellow are presented some alternative technologies.

4.3.1 *InterPlanetary File System*

In IPFS information is represented using a Merkle Directed Acyclic Graph (MDAG), which provides tamper-resistance properties to the data. The MDAG is a self-verified data structure which ensures the integrity of the data, since the identifier of each block (CID) is based not only on its content, but also on the CID of all the predecessor blocks. This means that if the content of a single block is changed, all the ascendants blocks are affected, therefore creating a brand new MDAG identified by a different CID. This is the reason why data on IPFS is immutable, it cannot be modified because it will no longer be associated with the same CID. If data is to be modified a new MDAG is published on the network.

After splitting the file, the blocks are stored on a node or scattered across multiple nodes. In IPFS each node stores only a small part of the network, unlike DLT where every node stores the entire ledger. This strategy reduces the redundancy in the network and also helps reducing the bandwidth used. Besides that IPFS uses a DHT for content discovery, which ensures fault-tolerance, scalability and above all decentralization. This table

maps CIDs to the nodes containing the respective blocks. Apart from content discovery the DHT is also used for *routing*, as it also maps nodes to their location, which needs to be queried in order to get the data.

Although being characterized by immutable storage IPFS is not append-only like DLT, as it allows for the removal of data through its garbage collection mechanism. When files are published in the network they are often pinned to the nodes in order to ensure persistency. If the data is not pinned to a node it will eventually be erased by the garbage collector. This mechanism is by default only activated when the storage of a node is near its limit, but it can also be run manually after unpinning a file or a cron-job can be used to perform garbage collection between a selected time period, ensuring that unpinned files are deleted relatively quickly. Even though garbage collection ensures removal of data on a specific node it cannot guarantee that the data has been removed from the entire network, since any other node can be hosting the same data and therefore it will remain available until every single copy is garbage collected. This may not ensure full compliance with the GDPR right to be forgotten, as the network cannot ensure that the data has been erased, even though it is still closer to compliance when compared to DLT where no data can be removed.

On top of IPFS there is also a decentralized name system, the **InterPlanetary Name System** (**IPNS**) [64], which allows for the creation and management of mutable links to content on the IPFS network. Given the immutable nature of data the IPNS provides a way to dynamically reference different IPFS content under a fixed name. An IPNS name can be viewed as a pointer that can be updated to reference different CIDs.

This decentralized name system is based on Self-Certifying File Systems (SFS) and Public Key Cryptography, so each node manages asymmetric key pairs. A name in the IPNS is represented as the *hash* of the public key and is associated with a signed record that contains the CID of the content referenced. The owner of the private key has control over the name-space, and only him can publish records, as each record is signed by the private key to ensure authenticity. IPNS records are stored on the IPFS DHT, alongside the other entries used for content discovery and routing.

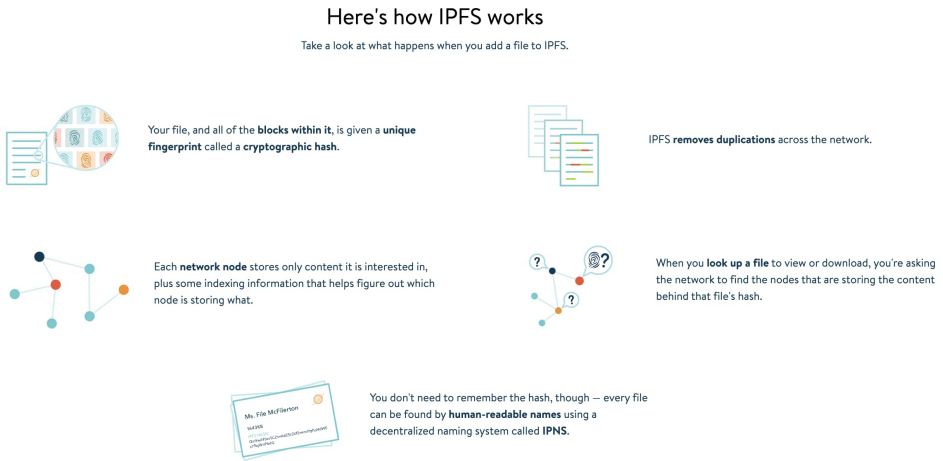


Figure 13: How IPFS works [65]

In the context of SSI the IPNS provides a suitable system to implement a DPKI given its decentralized features and ability to ensure integrity and authenticity of data, through the use of MDAGs and public key cryptography to sign IPNS records. It allows for the creation of a decentralized namespace, uniquely identified by the hash of a public key that can be used to reference arbitrary content on IPFS, in this case the different versions of a DID Document. The namespace is under full control of the owner of the private key and the whole system does not rely on any kind of centralized authority for the decision making process, thus fulfilling one of the key requirements of SSI. In fact IPNS is used as the supporting infrastructure for the **IPID DID Method** [66], so it should definitively be considered as an alternatives to DLT.

4.3.2 *GNUnet*

GNUnet [67, 68] is a decentralized network for anonymous and censorship-resistant file sharing, that can be used as a building block for secure decentralised applications. Its network topology is of a *Mesh Network* and like IPFS it uses a DHT, that combined with additional encryption mechanisms allow to anonymously publish and resolve content from the network. The GNUnet is made up of multiple nodes, and each one manages asymmetric cryptographic key pairs that are used both for node identification, authentication of the published data and encrypting communications.

In GNUnet content is encoded using the ECRS [69] (Encoding for Censorship-Resistant Sharing) format. When files are published in the network they are split in multiple 1 Kb blocks, uniquely identified by their RIPEMD-160 *hash*. GNUnet uses three different types of blocks to represent data, each one with its specific purpose. *Data Blocks* (DBlocks) are used to store the actual content of the file, and then *Indirection Blocks* (IBlocks) are used to link multiple DBlocks by storing their hash value. Each IBlocks may contain up to 25 hashes of DBlocks, along with CRC32 checksum and a *superhash*. This structure create a Merkle Tree that is then topped by a *Root Block* (RBlock) containing metadata about the file as well as the *hash* of the root IBlock.

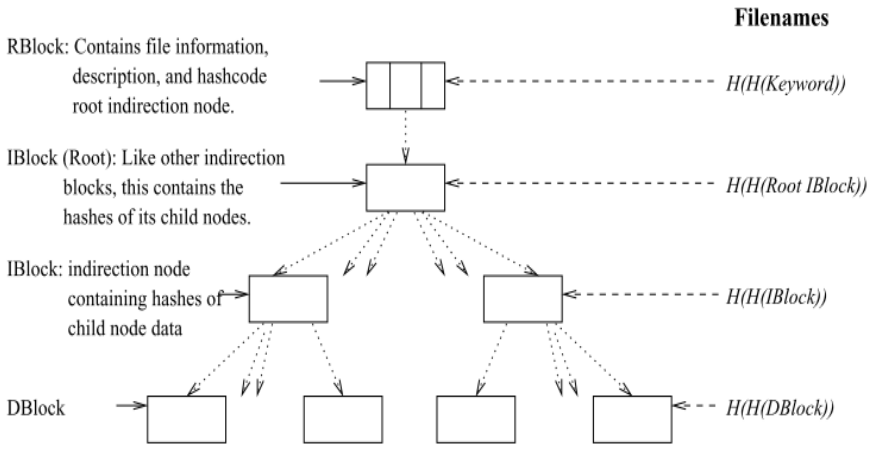


Figure 14: GNUnet File encoding [70]

Each block is then encrypted with a symmetric key derived from its *hash* value using a HKDF. All blocks are encrypted so its content is hidden from nodes storing or transmitting them, furthermore the encrypted blocks are stored on the DHT using the *double-hash* (*H(H(Block))*) as filename, so other nodes cannot guess the *hash* value of the original encrypted content.

GNUnet also defines a protocol for anonymous data transfer known as GAP (GNUnet Anonymity Protocol) [71]. This protocol defines formats and security mechanisms for queries and replies used for content resolution. In addiction to content encryption the GNUnet also encrypts queries to protect the identity of the peer sending the query and to ensure that other peers cannot discover the content of the query or the data it refers to. Besides that users can specify an anonymity level when publishing or searching content in the network, that defines how much cover traffic a node must use in order to stay anonymous.

On top of that GNUnet also defines a *ranking system* where every node maintains a value called *credit* for every other node it knows. Nodes monitor each other for available bandwidth and availability as well as for compliance with the network protocols. High availability and bandwidth will result in an increase in the ranking while malicious behaviour will decrease it. Also when peers performs queries their ranking is decreased, and when they provide valid replies it is increased. This mechanism is used mostly to mitigate DDOS attacks and prevent users from using too much of the network's resources without also contributing with their own. Nodes with low ranking are considered untrusted an their queries are only answered if there is excess bandwidth available.

GNUnet also includes the **GNU Name System**(**GNS**)[72, 73], a privacy-preserving decentralized name system that provides secure name resolution, query privacy, censorship resistance and also tries to be fully compatible with DNS. It is built on top of the GNUnet so some of it's privacy features, like query encryption and censorship resistance of the data, are inherited from the GNUnet itself.

The GNS allows to map names to GNS records that are cryptographically signed for authentication and integrity. Users manage multiple *zones*, each one defined by an asymmetric key pair and the set of GNS records associated with it. Records in a *zone* are stored on a local database under the control of the user and public records are made available to other peers through the GNUnet DHT. As any other block of data GNS records are encrypted before being published on the DHT, which allows for secure name resolution.

The GNS also provides some more advanced features like zone delegation and revocation. Delegation is performed using a specific GNS record, the PKEY record, that is stored on the DHT and ensures secure delegation of a *zone*. The validity of the delegation is ensured by a cryptographic signature and an expiration value. When it comes to *zone* revocation each node in the GNS maintains a local revocation list for *zone keys*. In order to revoke a *zone* a user must create and send a revocation message that is then flooded in the network. Each node that receives a valid revocation message adds it to the local revocation list and forwards the message to all its neighbours. The revocation message must include a *Proof of Work* (PoW) in order to prevent flooding attacks on the network. The revocation message must also be signed by the *zone* private key, so it must be calculated and stored beforehand, and when the key is compromised or lost the message should then be sent. So when resolving content in the GNS zones are checked against a revocation list to see if the zone key has been revoked,

if the key has been revoked content resolution fails.

The GNS is the current infrastructure used by **re:claimID** [45], a SSI system that combines the privacy-preserving mechanisms of the GNS with Attribute Base Encryption (ABE) to provide a decentralized digital identity that is capable of selective disclosure of attributes. Although self sovereign **re:claimID** does not comply with the standards defined by the W3C for DIDs and VCs, nonetheless the GNS seems to be a suitable infrastructure to implement a DPKI using DIDs.

4.3.3 *Freenet*

Freenet [74] is a decentralized peer-to-peer platform for secure data storage. It was one of the first decentralized, anonymous and censorship resistant systems, which heavily influenced the design of future approaches such as the GNUnet. One of its main features is the ability to anonymise users who interact with the network. This is ensured through a routing algorithm where requests are relayed through multiple node in order to obfuscate the identity of the node requesting the data. Freenet is constructed as a network of multiple nodes structured as a *Small World Network*, in which every node just knows their nearest neighbours. Each node maintains his own data store and a routing table that maps node addresses to the content they hold.

When data needs to be retrieved the requesting node relays the request to their neighbours, to check if they store the content, and the request is passed from neighbour to neighbour until the data is found or the hop limit is reached. Each node in this chain has no way to know if the node before him is the original data requester or just another intermediary relaying the request, therefore ensuring anonymity of nodes requesting and storing content. When the content is found the node storing it sends it back to the original requester and nodes along the chain add a new entry to their routing tables associating the *hash* value of the content with the respective holder.

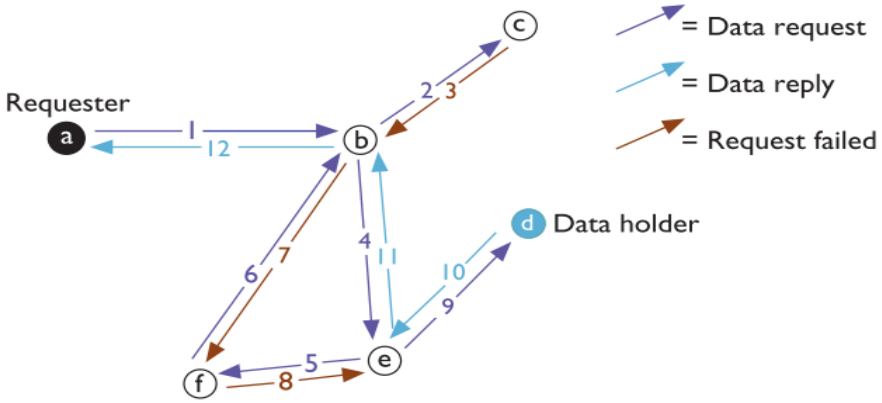


Figure 15: Freenet query routing [75]

Nodes relay requests based on the XOR distance between the *hash* values of content. So a node first send the request to the neighbour that contains the data with the closest XOR distance to the data it is searching for, if that node and its neighbours don't find the data, then the node send the request to the neighbour with the second

closest XOR distance and so on until the data is found. When adding data to Freenet the same mechanism is used, where the request is relayed from neighbour to neighbour based on the XOR distance and each node in this chain stores a copy of the content until the hop limit is reached.

Each file added to Freenet is split into multiple encrypted blocks that can be independently scattered across different nodes. Also redundant copies are created to allow for the data to be retrieved even if the original publisher node is not online. Redundancy of blocks is not just used to ensure data availability but sometimes blocks are moved or mirrored in order to prevent tracing or censorship from third parties. Besides that, encryption ensures that nodes storing or transmitting the blocks cannot disclose its content.

After inserting a file in Freenet it returns a *globally unique identifier* (GUID) key, that can then be used to later retrieve the file from the network. There are two main types of GUID keys in Freenet and each one of then is used for a different purpose. The *Content Hash Key* (CHK) is used to uniquely identify each piece of data and it is represented as the SHA-256 *hash* of the encrypted content. This ensures the integrity of each piece of data as other nodes have a way to verify that the data they are forwarding has not been tampered with, by hashing it and comparing it with the CHK. The *hash* of the content is then combined with the decryption key that deciphers the file to form the complete CHKs used to retrieve the data.

Besides that there are also *Signed Subspace Keys* (SSKs) which provide a personal namespace that any node in the network can read but only the node controlling the namespace can update. These keys are based on public key cryptography so when a node creates its namespace it first generates a public-private key pair, used to identify the node and sign namespace records, as well as an symmetric key used to encrypt data. The namespace is identified through the *hash* of the public key and symmetry key, and the private key is used to sign inserted records, so other nodes can ensure the authenticity and integrity of a signed-subspace file before storing it. Signed-subspace files do not store content directly but instead reference CHKs of data stored in Freenet. Because of that the namespace can be easily updated by creating a new referent to a different CHK representing a new version of the content. Due to the updateable nature of the namespace Freenet also provides *Updateable Subspace Keys* (USKs), a subtype of SSKs that abstracts users from the process of retrieving the most up to date version of the content associated with a SSK.

SSKs create a unique verifiable identity on Freenet that is associated with a namespace of records where users can publish content. Moreover each piece of data is encrypted and signed to ensure privacy, integrity and authenticity. These properties make Freenet a well qualified system to implement a DID Method, thus serving as an alternative to DLT.

Although there are some drawback related to Freenet, first the fact that it does not ensure the availability of stored data. Content that is not frequently accessed may be deleted from nodes without the publisher knowledge, in order to free space in node's storage. Also Freenet has no mechanism to explicitly erase data and therefore no way to ensure the right of erasure define in the GDPR.

4.3.4 *Dat Protocol*

The **Dat Protocol** [76], recently renamed **Hypercore Protocol**, is a distributed peer-to-peer protocol for secure data sharing, archival and versioning. It is heavily influence on BitTorrent and Git, and although intended for management and sharing of large dynamic data sets on decentralized networks, it can also be used for offline sharing on local networks or to create encrypted backups of data.

Dat does not use content addressing like IPFS or the other systems presented above, instead it implements a public key addressing schema, where each Dat repository is associated with a unique identifier derived from a public key. This ensures that the identifier will always remains the same even if data is changed overtime. Some of the main features of Dat include decentralized storage, encrypted data sharing and versioning. Users store their files locally, under their control, and make them available to the network via the unique identifier derived from the public key. Files are exchanged peer-to-peer without the need of any centralized authority and content is encrypted in order to guarantee privacy. When content is shared the receiving nodes have the ability to re-host the content improving the availability of the data in the network and ensuring that data is still reachable even if the publisher node is offline.

Integrity of data is also ensured, as file are split into multiple chunks that are *hashed* and then used as leaf nodes to create a Merkle Tree. This structure allows other peers to verify that data has not been tampered with, and an additional signature on the root node of the tree guarantees the authenticity of data. When new content is added to the repository or a changes is performed, new chunks are added the tree is reconstructed and the root hash is re-signed.

The public-private key pair associated with each Dat repository is also used to enforce read and write permissions to the files. Files are encrypted using the public key, so this key is needed in order to access the content of the files. Besides that only the controller of the private key has the ability to update files, given that a signature is required to publish changes or add new content to the repository. Since Dat uses the public key to encrypt the content, both public and private key should be kept secret. Because of that Dat uses a third key know as *discovery key*, the BLAKE2b hash of the public key, to discover which peers in the network store the content without exposing the public key.

The unique thing about Dat is that it stores data in binary append-only registers, known as *Feeds*. For each repository Dat creates two *feeds*, the *content feed*, where the actual content of the files is stored, and the *metadata feed* used to track all the changes performed in the repository. The content of each one of the *feeds* is cryptographically hashed and signed so other peers downloading and re-hosting the content can verify the change history and trust the current state of the repository.

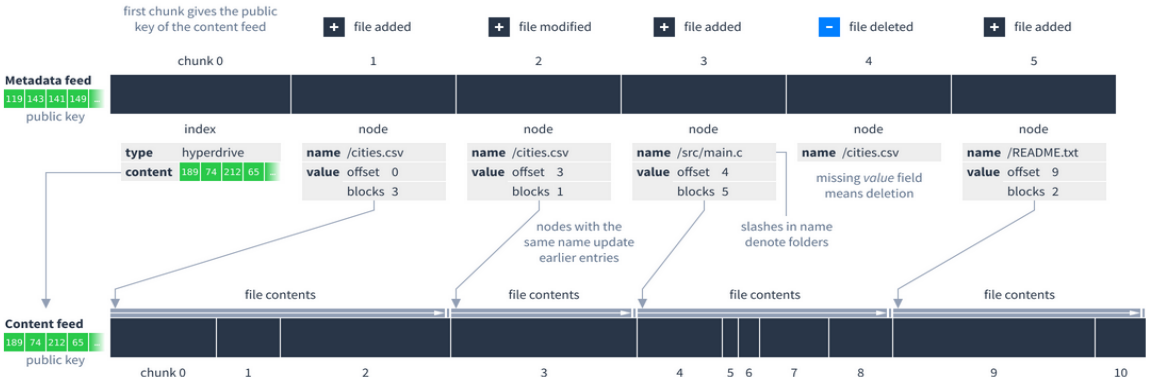


Figure 16: Dat Content and Metadata *Feeds* [77]

Besides keeping track of changes the metadata feed also store pointers to data chunks on the content feed, which provides better efficiency when traversing the log, also allowing for partial download and hosting, as peers can specify an offset and download just a subset of the content in the repository. By using these append-only registers Dat creates a built-in versioning system that keeps an accurate and verifiable history of all the changes performed in the repository.

Although Dat was created as an infrastructure for distributed sharing of large data sets, it can also be used as a building block for decentralized applications. In fact its decentralized nature combined with the tamper-proof and encryption mechanisms make Dat a suitable system to replace DLT as decentralized identity register in SSI systems. Furthermore other non-DLT infrastructures such as the IPFS and Gnunet have to build name systems on top of their network in order access content via a persistent unique identifier. This create and additional step when retrieving content from the network which impacts the overall performance. Due to its public key addressing schema Dat is able to generate persistent identifiers that point to dynamic content without needing an additional name system.

4.4 NON-DLT SOLUTIONS

Although non-DLT SSI is not a vastly explored concept in the field of digital identity, there are already a number of implementations and tools developed. This section provides an brief overview over some identity management systems that are supported by non-DLT. Besides fully developed SSI systems, some non-DLT DID methods are also presented, since in view of the architecture defined by W3C they represent the bottom layer of the SSI stack that supports the DPKI and provides trust for the whole ecosystem.

4.4.1 *DID Methods*

Given that the focus of this thesis is on non-DLT SSI and alternative decentralized infrastructures that can be used in the implementations of DID Methods, as seen in section 4.3, below are presented some public DID Methods that do not rely on DLT as their source of trust. Nonetheless SSI also defines the concept of pairwise DIDs intended for private relationship. Since they are only relevant to the peers taking action in that relationship, pairwise DIDs are not stored in any publicly accessible database or registry, instead they are stored locally and exchanged peer-to-peer so no external entity can resolve them. Given that the source of trust for these methods is based on the public-private key pair used to generate the DID and initial DID Document and not on any DLT, some pairwise DIDs are also included in this overview.

Public DID Methods

The **IPID DID Method**[66] uses the Interplanetary File System (IPFS) to create and manage DIDs, more specifically it uses Interplanetary Linked Data (IPLD) to represent DID Documents and the Interplanetary Name System (IPNS) to manage and resolve them. The IPLD is an abstract data model that enables linking of data via cryptographic hashes, providing integrity protection. Also before being stored IPLD is serialized into Concise Binary Object Representation (CBOR). The IPFS is a content-addressable network in which data is associated with a unique identifier known as Content Identifiers (CIDs), derived from the *hash* value of the data, that is used to retrieve content from the network. Given that the CID is directly related to the content of a file, every time the file changes the CID will change as well. So in order to provide a persistent identifier the IPNS is used to create and manage links to IPFS content. A name in the IPNS is represented as the hash of a public key, and it's always associated with a record that contains data and is cryptographically signed by the private key. The IPNS record stores the CID of a file, and every time the file changes a new CID is calculated and the record is updated, associating the new CID to the IPNS name. This ensures that the latest version of the DID Document is always retrievable via IPNS name.

In the IPID method the DID is associated to the DID Document by registering the CID associated with the IPNS public key on a Distributed Hash Table (DHT), and it's method specific identifier is the IPNS name, which can be used by other peers to resolve the DID Document from the IPFS network.

This method can also supports pairwise interactions, if used on a private IPFS networks, although it was mainly conceived to be a public DID method that can be used as a trust anchor for SSI systems as any other DLT based method. It's main advantages when compared with other DLT based methods are better scalability and cost-efficiency when it comes to creating and managing DIDs and DID Documents.

Other DID methods use a hybrid approach, like the **Jolocom** [78] and **Element DID methods** [79] that combine the Ethereum blockchain with IPFS. The Jolocom method uses IPFS as storage for DID Documents, and then a Ethereum smart-contract maps the DID to the IPFS address of the corresponding document. The Element method is an implementation of the Sidetree protocol that also uses Ethereum on top of IPFS.

These methods further enforces that a pure DLT approach might not be the best solution to this problem since the high latency of DLT transactions and heavy data replication make scalability almost impossible.

Pairwise DID Methods

The **Peer DID** Method [80] is a DLT independent DID method that was designed for private peer-to-peer relationships and offline use. The argument of it's creators is that DIDs that are used in private relationships should not be stored on DLTs, distributed file-system or other publicly available repository as it posses a privacy and security risk. Instead Peer DIDs and DID Documents are kept locally on wallets together with credentials and can only be resolved by the relationship's participants, and no other external entity. This greatly reduces concerns on sharing private data and compliance with privacy regulations since a very limited numbers of peers can resolve the DID Document.

To create a DID with this method a user must generate an asymmetric key pair that is used to create the *genesis* DID Document, an initial DID Document without any endpoint or ID field, just containing the *inception key*. The method specific identifier of the DID syntax is then created by calculating the SHA256 of the *genesis* DID Document. This is the most common way of creating a peer DID, although the public key can also be directly encoded to create the method specific identifier, in this case the Peer DID is associated with an empty document. The root of trust for Peer DIDs is based on the entropy of the inception key. This key must be unique for each DID and never reused for any other purposes since it is needed to authorize the exchange of DID Documents in every relationship and to prove that its holder is indeed the entity that created the DID.

Peer DIDs support updates to the DID Document allowing holders to add, delete and rotate keys as well as endpoints. Any update performed to the Document is registered in a *Delta*, a piece of JSON data containing the encoded fragment that has been updated, a signature of the fragment, a key reference and a timestamp. *Deltas* are used to ensure the authenticity and integrity of changes performed to the DID Document. Besides that Peer DIDs are compared using its log of *Deltas*, instead of the semantic or syntactic representation. If two Peer DIDs are associated with the same log of *Deltas* they are considered to be the same, independently of the representation. Considering that each Peer DID is associated with a log of *Deltas* there must be a storage mechanism to keeps the history of all the Deltas. The storage mechanism can range from a file to a database and it serves as the decentralised identity register for this DID method.

Remember that Peer DIDs should only be used for private peer-to-peer interactions as it only supports Pairwise or N-wise relationship, where N is relatively small. Even in that case they should not be reused across relationships in order to avoid correlation. For public interactions that need to be verified by third party entities, a public DID should be used instead.

There are also other pairwise DID methods that follow a more simplistic but ledger independent way to create and manage DIDs. This is the case of the **Key DID Methods** [81], a non-registry Method that directly encodes a public key as the DID value. DID Documents associated with a Key DID do not need to be stored on an identity

registry, given that they are derived from the public key in a deterministic way. With the DID or the public key value the document can be reconstructed. Since Key DIDs are generated in a deterministic way they do not support updates to the DID Document, neither deactivation of a DID, making it impossible to recover from any security compromise. With this in mind Key DIDs should only be used for single and short term relationships that last less than a week.

4.4.2 *SSI Systems*

Besides DID methods there are already entire SSI management system developed with non DLT technology. Despite being self sovereign, some of the SSI systems presented bellow may not fully comply with the standards for DIDs and VCs, still they follow a very similar architecture, also ensuring the final goal of SSI that is allowing users to take back control over their data and digital identities.

IRMA

IRMA [48] is a decentralized and attribute-based platform that allows users to control their identity and use it to authenticate in multiple service providers without the need of a central authority. It is based on the Idemix attribute-based credential scheme (ABC) [82], which allows for selectively disclose of attributes in a privacy preserving and unlinkable manner.

In this scheme credentials are used as containers for multiple attributes, which are then disclosed to service provides in order to authenticate or prove some properties about a user in a verifiable way. Idemix Credentials contain one or more attribute, being that the first attribute of all credentials is a special attribute representing the user's secret key. This secrete key is a random integer generated on the first interaction with the IRMA app, and it is used to identify the user and link credentials to him. The secret key attribute is never disclosed to any third party, not even to the issuer in the process of issuing the credential. Besides attribute Idemix credentials also include a cryptographic signature from the issuer, used to ensure the integrity and authenticity of the attributes, and well as an expiration date.

The Idemix credential schema follows a very similar architecture to the one defined by W3C for SSI, where a trusted third party called *Issuer* generates and digitally signs credentials using his private key which can then be verified by any other party who knows the corresponding public key. Hence the public keys of an IRMA Issuer is made available to all the participants, via the Scheme manager, in order to make credential validation possible. Credentials and attributes are stored locally on the IRMA mobile app and shared directly with service provider, also known as *Verifiers*. This allows users to take back control over their personal data and creates a secure decentralized ecosystem for data sharing. Furthermore IRMA puts a big effort into minimizing the disclosures of personal data, as users are given the ability to selectively disclose any subset of attributes contained in a credential, by revealing only the ones that are relevant for the interaction and hiding the others that are not needed. This disclosure is done in a zero-knowledge way, where the user shares the disclosed attributes as well as a *proof of knowledge* which allows the verifier to guarantee that the users knows and has a valid signature from

the issuer for the attributes that are being disclosed as well as for the ones that are hidden. Also the disclosure proof, which includes the disclosed attributes and the proof of knowledge, is unlinkable meaning that the verifier cannot distinguish between two different credentials with the same set of disclosed attributes.

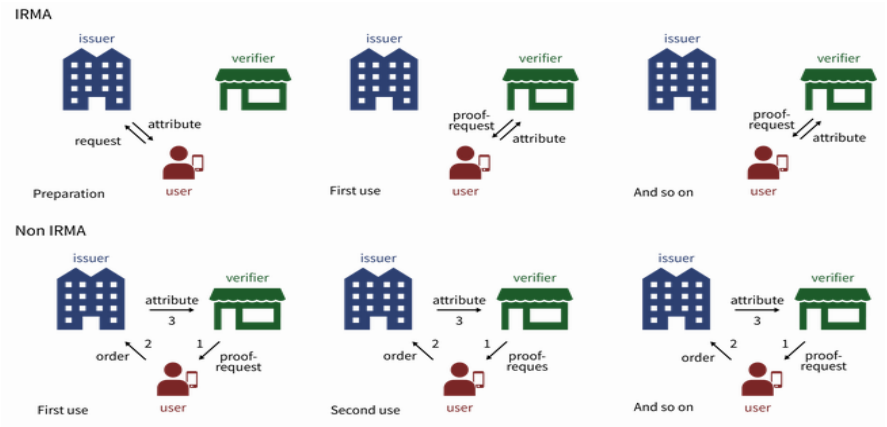


Figure 17: IRMA attribute exchange [83]

IRMA also provides mechanisms for credential revocation. Revocation is used to ensure issuers can invalidate or void previously issued credentials when their attributes become outdated or are no longer accurate. The revocation mechanism used by IRMA is based on a type of cryptographic accumulator that allow revocation to be performed in a zero-knowledge way [84]. The accumulator representing the revocation list for an IRMA issuer must be available to all participants, and when the state of the accumulator changes a revocation update message is sent to all the participants to notify the change.

Besides ABC, IRMA also uses attribute-based signatures (ABS), a type of digital signature that allows the signer to attach some of his attributes to the actual signature. These attached attributes are then accessible to anyone verifying the signature.

Another focal component of the IRMA ecosystem is the *Keyshare* server, a centralized server used to store part of the user's secret key and for revocation purposes. The user secret keys is split and stored on the *Keyshare* server as a security measure, since the mobile phone where the IRMA app is hosted, can easily be stolen or rooted to access the secret key. When the user first interact with the *Keyshare* server it defines a PIN code, that must be provided in every interaction with the *Keyshare* server so it cooperates with his half of the secret key. This code should not be forgotten as there is no mechanism to change it, so forgetting your PIN code means losing control of your attributes. The *Keyshare* server is mainly used in the process of issuing and disclosing attributes, however it does not know which attributes are being disclosed or to whom they are being disclosed. It just provides his half of the key after validating the PIN code and the rest of the interaction is performed directly between the IRMA user and the relying party. This server also has a web page associated with it, where users can remotely block their IRMA app and revoke their attributes in case their mobile device gets lost or stolen. Besides storing part of the user's secret key and allowing for revocation the *Keyshare* server also keeps a record

of IRMA transactions and PIN code entry attempts for security purposes, although it just stores transaction timestamps to ensure the privacy of the exchanged attributes.

Although more secure, the use of this centralized server create a single point of failure in the IRMA architecture which significantly affects its decentralized nature. Hence if the *Keyshare* server ever becomes unavailable no user in the IRMA ecosystem will be able to issue or disclose attribute.

IRMA has been around for quite some time and provides a DLT independent SSI system that has been mainly used in Netherlands to issue government credentials. Although it is open-source and GDPR-compliant it does not conform with the W3C standards for DIDs and VC. Nonetheless it follows a very similar architecture where users control their own identities that are expressed via attributes contained in digitally signed credentials. These credentials are issued by trusted third parties and can be verified be anyone who knows the public key. One major difference is the fact that IRMA partially relies on a centralized server for revocation and key storage.

re:claimID

re:claimID [45] is a decentralized identity management system build on top of the GNU Name System(GNS). It enables users to securely manage and control their digital identities by storing attributes in user-owned *namespaces* on the GNS. Each namespace contains a name-value mapping of records that is cryptographically linked to a public-private key pair controlled by the user. Before being published in the namesystem attributes are encrypted using ABE, which allows users to enforce access control on their personal data. This is achieved by generating a individual access key for each requesting parties that the users can revoke or update at any time. Requesting parties retrieve encrypted data from the *namesystem* and can only decrypt the attributes that the users has granted permission during the generation of their access key. After obtaining the access key they can retrieve information from the *namesystem* even if the user if offline.

Another big advantages of this implementation is that it is fully compatible with the OpenID Connect protocol (OIDC). Web sites and users can use the OIDC protocol to request authorization and exchange access keys to re:claimID attributes. Furthermore it also allows for the same Single Sign-On (SSO) authorization flow as the one provided by federated identity provider such as Facebook or Google, ensuring easy integration with web application.

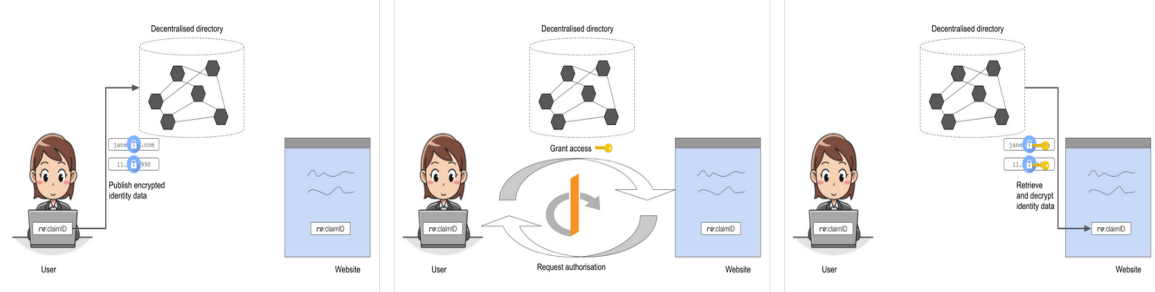


Figure 18: re:claimID workflow [85]

Other security properties of re:claimID are inherited from the GNS, such as the integrity protection, authenticity and confidentiality of data. GNS records are signed using a key derived from the namespace private key and the DHT has a signature verification mechanism that only stores and returns valid data. Records are also encrypted which ensures the privacy and confidentiality of the stored data, and queries are protected via a query privacy mechanism. The GNS relies on a R5N DHT [86], which randomly replicates data across multiple node to ensures an high availability, resilience and censorship-resistance. On the other hand the fact that data cannot be easily removed from the GNS, as records can still remain cached around the network, can make the removal of attributes a difficult task. Because of that re:claimID enforces a revocation mechanism to ensure that requesting parties that were previously authorized to access the attribute cannot do it anymore after it has been marked as removed. This is done through the ABE policy, which can be used at any time to revoke access to a a subset attributes. The attribute could still be stored in the namesystem, but no requesting party can decipher it.

So the main idea behind re:claimID, is combining a decentralized name system with an ABE policy, to create a self-sovereign data sharing system. The namesystem provides decentralized storage with integrity and authenticity guarantees, and the ABE policy allows users to enforce access control over their attributes. Re:claimID identities are initially self-asserted nonetheless it can be used to share third party attested attributes.

Identity Manager

The **Identity Manager** (IDM) [87] is an open source identity wallet that enables users to manage multiple digital identities. It is fully compliant with the standards on DID and VC and it also uses DID Auth to prove control over a DID. Its main goal it to provide an identity wallet that is able to support multiple identities and DID Methods, given that in the current landscape users may have to use different wallets for different DID Method and their associated identities. This can be a major usability problem, as it increases the complexity of handling DIDs from different methods, so IDM tries to answer this problem with their approach.

Besides DIDs and VCs the IDM also manages the devices and applications that interact with the different identities. This enables users to revoke a device if it ever becomes compromised and revoke sessions with application. All users' data is stored encrypted inside a wallet that also handles the signing and verification of cryptographic signatures as well as the authentication process (DID Auth).

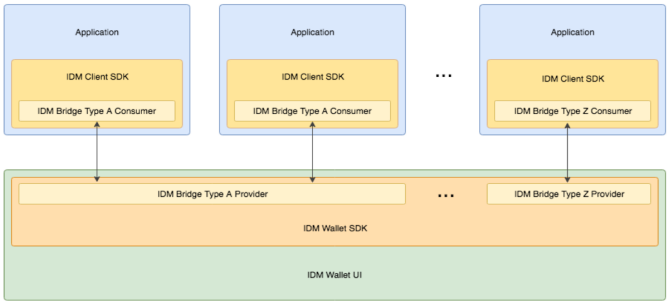


Figure 19: IDM High Level Architecture [88]

From an architectural point of view IDM is composed of four major components, the Wallet, Wallet UI, Client and Bridge as shown in Figure 19. The wallet is the main component and it is used to securely store most of the user's data such as the DIDs, VCs and devices and applications lists. This component is also responsible for the creation of DIDs, although importing existing ones is also possible. One major piece of data that the IDM wallet does not store are DID Master Keys. Given that this key gives complete control over a DID, IDM chooses not to store it for security reasons, and advice's its users to keep it backed up in a paper format. IDM also provides a secret sharing schema that allows users to split the DID Master Key into multiple pieces and share them with a group of trustees, enabling them to latter reconstruct the Master Key from some of the shares if needed.

The Client component is meant to be integrated in applications and it is used to interact with the IDM Wallet. Though this interaction is mediated by the IDM Bridge, which ensures the communication between these two components. Each Bridge can support different transport protocols to assure that messages can be exchanged with applications in different contexts, either running on a web browser or on a mobile device.

IDM can support both DLT and non-DLT DID Methods, but for now it only supports the IPID DID Method, which is non-DLT. However given that it's main focus is on coupling multiple DID Methods in a single wallet, it will likely include non-DLT methods in the future.

Peergos

Another project worth mention is **Peergos** [89] a decentralized protocol and storage platform where users can securely store and share their data with other peers. It was designed as a self sovereign alternative to DropBox or Google Drive, and not as a identity management system, nonetheless it shares the same goal of given back control and ownership of data to the users. Peergos uses the IPFS as the storage platform and the cryptree protocol [90] to enforce access control over the data. This protocol allows for a fine grain access control where users can grant access to only a subset of their data. Before adding the data to IPFS, Peergos encrypts files in 5 MB block that are then split into smaller fragments and uploaded, this is done to ensure that anybody hosting the content on IPFS cannot read it. Besides that each user manages multiple cryptography key pairs not only to provide access to their data, but also to decrypt its own root directory and sign data.

Its architecture is independent from any central certificate authority and can be completely self-hosted, although it is not fully decentralized since it relies on a centralized PKI server, the *Corenode*, to manage the uniqueness of usernames and store users public keys. Even so this centralized server is mirrored in other Peergos servers, to ensure an high resilience to attacks, if the PKI server ever becomes compromised or unavailable the network can easily move to another one without losing any of the data. It also ensure privacy on public key lookups and provides better scalability. Besides the public keys of each user no other key is stored on any server, instead they are derived from the username and password using the *scrypt* key-derivation algorithm [91]. This algorithm outputs the identity key pairs as well as an encryption key used to decrypt the user's root directory.

Peergos puts great focus on privacy and security, however it does not guarantee the anonymity of users, still it aims to integrate onion routing [92] to enforce this property in the future. Its documentation also mentions the possibility of integrating a blockchain to ensure a more decentralized architecture, but for now it is still DLT independent.

4.5 ISSUES AND LIMITATIONS

Decentralized peer-to-peer networks provide a good alternative for DLTs in the context of SSI, however there are a number or issues and limitations that need to be addressed in order to support SSI in its full potential. This section presents some of these issues which are mainly related to the security, anonymity and latency of the networks as well as some underlying network mechanisms that may make some SSI features impossible.

4.5.1 *Security*

Despite using a number of security mechanisms to ensure the anonymity and confidentiality of users and data, decentralized networks are vulnerable and susceptible to attacks like any other system. There are design flaws in these systems that can be exploited to compromise the anonymization protocol, reputation system or availability of data.

The most prominent attacks on decentralized networks include the Sybil and Eclipse attacks which affect almost all decentralized networks on a conceptual level [93]. In the Sybil attack a single rogue node creates a huge number of identifiers to try to influence the reputation system and gain power in the network. This can be a significant problem for public network where users can create as many identifier as they want, allowing a single entity or small group of entities to take control of a significant part of the network. A Sybil attack can also be used to censor content on the network or make it unavailable. Since the node identifier is used in the routing protocol, generating a large amount of identifier in a specific key range would result in the rogue identifiers taking the place of valid peers in the routing table, giving control of content to the attacker and enabling him to censor information.

The Sybil attack can also be used to perform other more complex attacks, like the Eclipse attack. In this scenario the attacker tries to isolate the victim node from the rest of network by infecting its local routing table

with rogue peers controlled by him, ensuring that all the traffic that comes and goes from the victim node passes through the rogue node as well. To do that the attacker sets up a Sybil attack to generate identifiers close to the victims range until its routing table is completely made up of malicious peers, and thus isolates the victim from the rest of the network.

Most public decentralized networks are conceptually vulnerable to these two attacks, its a matter of how many resources are needed for the attack to be successful, and if it can be done in a reasonable amount of time. Despite enforcing specific mechanisms and strategies to ensure that these attacks cannot be performed with ease or by a single machine, there are researches that demonstrate some decentralized networks are still vulnerable to similar attacks. In [94, 95, 96] the authors show that Freenet is vulnerable to the Routing Table Insertion (RTI) and Traceback attacks. These attacks use a similar strategy to the Sybil and Eclipse attack to compromise the anonymity of the system. With the RTI attack an attacker can insert himself in the victims routing table, and with this ability other attacks on the anonymity of the system can be performed. One of such attacks is the Traceback attack which enables an attacker to determine which nodes have been used to route a specific message and who is the message originator. This attack starts by deploying some rogue nodes along the network and perform the RTI attack to become a neighbour node of the victim and monitor messages exchanged. To determine the group of nodes involved in the exchange of a message and its originator the attacker queries all of the victims neighbour to determine other nodes that have forward the message. By the end of this process it will have discovered all of the nodes involved in the exchange of the message and the originator node as well. In [96] the author also mentions a vulnerability on the GNUnet Bloom filters which can also be exploited to determine the group of nodes involved in the routing of a message.

4.5.2 *Latency*

There are few comparison studies on the performance of DLTs and decentralized peer-to-peer networks, however decentralized networks often provide better scalability and performance since they do not bear the overhead of a consensus protocol. In [97] the authors provide a comparison between IPFS and a blockchain based storage system, the Sia Skynet, in terms of request latency and network errors. In their research they were able to show that IPFS has significantly lower request latency and produces less errors than the DLT based solution. Also [98] shows that the high transaction latency associated with DLTs is mostly related to the transaction validation step.

Despite the underlying network providing better performance, DID resolution can be considerably slow on these networks. The high latency of DID resolution may not be related to the network latency itself but rather to the high overhead added by the name system. Since most of the decentralized networks are characterized by immutable data, a name system is often used to create a mutable link to reference data under a unique persistent identifier. More specifically this enables DID Documents to be updated and remain associated with the same DID. The IPFS and GNUnet are two examples of decentralized networks that rely on name systems to provide the ability to dynamically reference immutable content, however there are networks that do not follow this approach,

as it is the case of Dat. In this case the latency of DID resolution can be much lower given that there is no additional overhead of the name resolution.

4.5.3 *Data Availability and Erasure*

Another major issue is related to data availability. Unlike DLTs most decentralized networks do not enforce complete replication of data amongst all the network nodes, although some data replication is enforced. In DLT every node stores the entire ledger, so any of the nodes can be used to access a piece of information, thus ensuring the availability of all the information stored on the ledger. On decentralized networks data is stored on a few nodes and the level of replication differs based on the popularity or ratting of the content. This means that for a piece of data to be retrievable there must be at least one node in the network storing it, otherwise the data will be unreachable. The thing that makes this issue much worst is the fact that nodes often leave and join the network, making data availability uncertain.

This can be a concern when using these networks as a persistent storage infrastructure, however there are third party pinning services that can be used to ensure that data stays available at all times. Also in the context of SSI the DKMS Cloud agent could be used to host the network node ensuring the availability of DID Documents and revocation registries.

Erasure can also be a problem given that most decentralized networks do not provide mechanisms for data removal in the entire network. In IPFS data can be removed from a local node however there are no guarantees that other nodes in the networks are not still hosting the content, making it still available. Data can also be erased due to storage limitation on the nodes or due to is low popularity. This occurs in GNUnet and Freenet where data gets erased from the network if it is not requested for a period of time.

Despite providing the ability to erase data this is not done due to the user desire to erase the information but rather to network mechanisms. Most networks enforce these data erasure mechanisms, however this is not enough to comply with the right of erasure defined by the GDPR, given that these they cannot ensure that the data has been completely erased from the network. Nonetheless this seems to have an easier solution on decentralized networks that in DLTs, given that they are not append only structures by nature. A protocol for explicit data removal could be build on top of the existing infrastructure, and in [99] such a protocol is proposed for the IPFS. In this research the authors define a protocol to delegate erasure requests between the IPFS nodes in order to guarantee that the information is completely erased from the network. It shows that a data erasure mechanism could be easily integrated in the existing infrastructures and would have a minimal overhead on the network performance. In GNUnet a similar mechanism could also be implemented given that the network already uses on a gossip protocol to share key revocation notices with all peers. Such a protocol could also be used to implement a data erasure mechanism in the network.

4.5.4 *DID Master Key Revocation and Rotation*

SSI is an identity management paradigm that heavily relies on public key cryptography, as identities are represented and controlled by cryptographic key pairs. That being said, SSI systems and the underlying identity registry, in this case decentralized names systems and networks, must provide mechanisms to recover from key lost and compromise. If not the identity owner will lose control over his identity and associated credentials in case the DID master key is lost or compromised by an attacker.

Key lost can be tacked using key recovery mechanisms, like Shamir Secret Sharing to split the master key amongst a group of trustees and reconstruct it afterwards if needed, however key compromise seems to be a harder problem to solve. The most common solutions for key compromise are key rotation and revocation, however due to the design choices of most decentralized name systems and network the rotation and revocation of the master key pair are either impossible or not supported. If the underlying system does not provide such mechanism an attacker that compromises the master key will be able to permanently impersonate the user, given that there is no way to mark the key or DID as compromised neither the identity owner has the ability to rotate such key.

Master key rotation is impossible in every networks and name systems mention in this thesis. This is due to the fact that the namespace ID, that is also used as Method Identifier for the DID, is derived from the master key pair. So rotating the master key would imply changing the namespace ID, thus resulting in a different DID and identity. Considering that key rotation cannot be used to solve key compromise in a non-DLT approach.

Since key rotation is impossible the only other option to tackle master key compromise is key revocation, yet this mechanism is only supported by few networks and namesystems. In fact from all the systems mentioned only the GNS allows for the revocation of the master keys. To do this it uses a revocation certificate that needs to be signed by the private master key and is then flooded through the network using a gossip protocol. The revocation certificate also needs to include a Pow which is used to make the revocation procedure deliberately hard, so malicious users cannot flood the network with revocation messages. Given that the revocation message requires a Pow which can take some time to compute the user is able to compute the revocation certificate beforehand and latter publish it if the key becomes compromised.

So key revocation seems to be the most prominent solution for master key compromise in non-DLT systems. Unlike key rotation which is impossible due to the system architecture, key revocation mechanisms can be integrated in these namesystems and networks. Nonetheless most non-DLT system still lack the ability to revoke the master key, and to provide a viable alternative to DLTs, which has similar levels of security and assurance, they must enforce this mechanisms.

5

A NON-DLT FRAMEWORK FOR SSI

This chapter introduces a non-DLT SSI framework that was developed in the context of this thesis. The main reason behind the development of this framework is the fact that there are few to no SSI frameworks that support non-DLT DID Methods, despite some of them already being adopted by the W3C Credentials Community Group as valid DID Methods [100].

In the first section of this chapter a high level overview of the framework is provided, mentioning its main features, infrastructures and specifications in which it is built. Then each of the framework modules is described and the architecture is presented. Finally the chapter ends with the implementation of a *proof of concept* application using the framework.

5.1 OVERVIEW

This SSI framework relies on non-DLT DID Methods only and provides the basic functionalities to create SSI agents and applications. It covers all the main SSI standards, presented in chapter 3, providing the ability to create and manage multiple decentralized identities and associated credentials as well as authenticate and securely communicate with other entities. The framework is merely experimental, nonetheless it aims to provide a complete SSI tool that enables decentralized identity management without relying on DLTs. Despite the nature of the supported DID Methods the framework is able to provides all the major features that every SSI actor needs to perform their duties in the SSI ecosystem. From creating and managing digital identities using DIDs, all the way to the issuance, verification and revocation of VCs, every major functionality is included.

Below the main feature of the framework are summarized:

- Enables the creation and management of multiple digital identities using different non-DLT DID Methods (public and pairwise).
- Allows updates on DID Documents in order to add and modify keys, authentication methods and service endpoint.
- Provides a key recovery mechanisms, based on the Shamir Secret Sharing algorithm, that can be used in case of DID Master Key lost.

- Allows the generation of cryptographic keys pairs as well as signing and verifying digital signature on Linked Data Documents.
- Provides an encrypted wallet where users can securely store their private information, such as credentials, identity data and private keys associated with DIDs.
- Supports protocols for authentication and secure communication that rely on DIDs and their associated cryptographic keys (DID Auth and DID Comm)
- Enables the creation and verification of VCs as well as Credential Schemas and Credentials Revocation Lists
- Supports revocation of credentials through the use of Credential Revocation Lists

To enable these functionalities the framework relies on several different specifications and infrastructures. Below the supporting infrastructures and specifications are summarize in order to better understand how the functionalities are supported. Figure 20 shows the different SSI standards and the corresponding infrastructure or specification used to implement it. Each of the framework modules will be discussed in detail in the next section, nonetheless as an introduction to the framework an overview of the supporting infrastructures and implemented specifications is provided here.

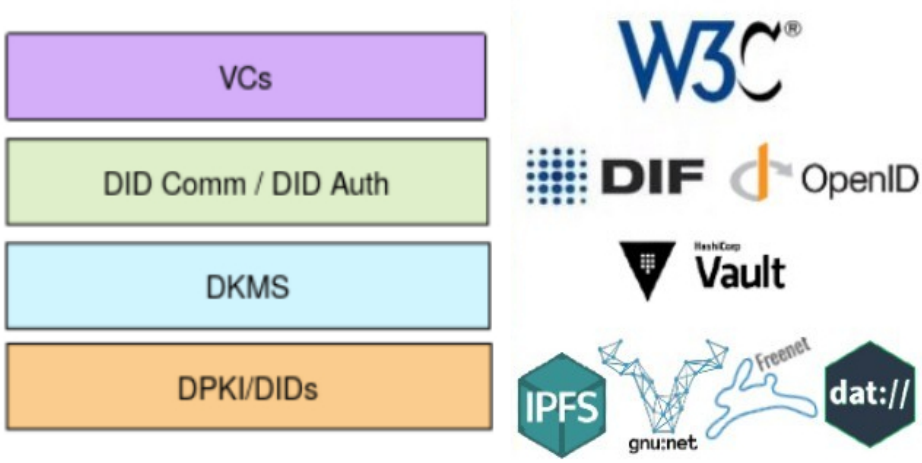


Figure 20: SSI Standards and Framework Infrastructures

Given that the main focus of this thesis is on non-DLT SSI and due to the small number of non-DLT DID Methods available, the framework includes implementations of the DID standard over the different decentralized networks presented in section 4.3. For DIDs on the IPFS the framework follows the IPID DID Method specification defined in [66] and for the GNS, DAT and Freenet a reference implementation is provided. The reference implementations follow a similar approach to the IPID DID Method where the DID Document is published on the decentralized network under a namespace that is controlled and identified by a public-private key pair. The hash

of the public key is used to identify the namespace and to derive the *Method Specific Identifier* part of the DID syntax and the private key is used to authorize update to the namespace.

Besides public DIDs that are stored on decentralized networks the framework also supports pairwise DIDs for private and offline interactions. For these interactions the Peer DID Method and Key DID Method are supported.

When it comes to the DKMS standard the framework provides an identity wallet where users can securely store and manage their personal information, mainly DID private keys and VCs. This identity wallet is built on top of Hashicorp Vault [101], an encrypted storage platform commonly used to store cryptographic keys, certificates and other types of private information. The identity wallet is a focal component of the ecosystem, given that it enables identity owners to move further away from the centralized identity approach, by taking control of the storage and management of their private data.

The framework also includes protocols to handle secure communications and authentication based on DIDs. For DID Comm it follows the DIF specification defined in [35] to implement encrypted envelopes. And for DID Auth it provides an implementation of the DID SIOP Profile, an adaptation of the OpenID Connect SIOP flow that uses DIDs [26]. To initiate the authentication or communication process both protocols rely on service endpoints defined in the DID Documents, so identity owners must define these endpoint before trying interact with other entities in the ecosystem.

For credentials, the framework includes an implementation of the W3c specification for VCs, that enables the creation and verification of both VCS and VPs. Besides that it also supports the creation and management of Credential Schemas and Credential Revocation Lists. For the credential schemas it follows the specification for VC JSON Schemas defined in [102] and for the credentials revocation lists, it provides an implementation of the *Credential Status List 2017* [103] and *Revocation List 2020* [104].

5.2 MODULES

In the previous section a high level overview of the framework was provided mentioning the supporting infrastructures and implemented specifications. This section will dive deeper into each of the framework module by describing the main functionalities provided by each one of them.

The framework follows a very simple design being made of five major modules, representing the SSI standards. Figure 21 illustrates the framework design and highlights the main functionalities of the different modules.

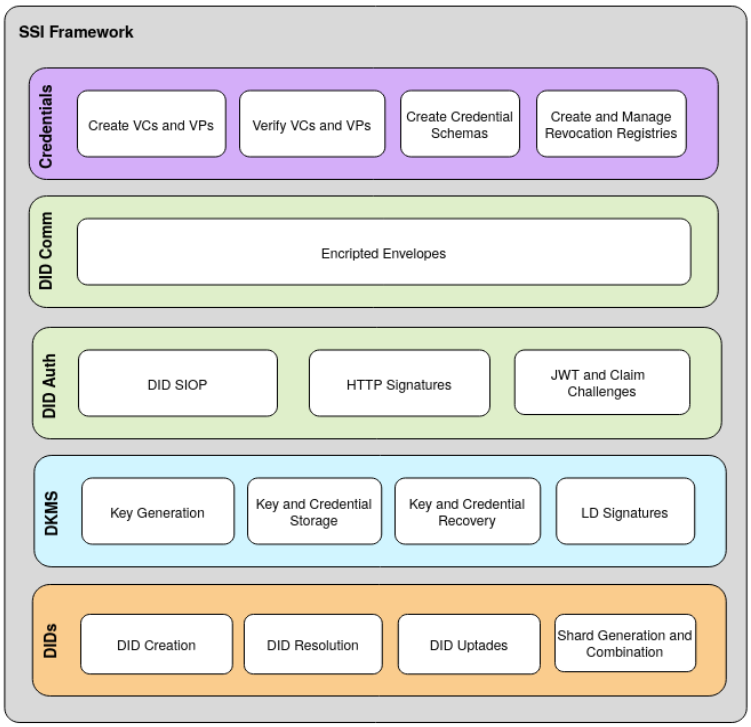


Figure 21: Framework Modules

DID Module

The DID module provides functionalities for the creation and management of DIDs and DID Documents. It is a central part of the framework given that it implements the DPKI which serves as the trust infrastructure for the whole ecosystem. It allows entities to create cryptographically asserted identities that are verifiable, completely self managed and independent from identity providers. This module supports multiple DID methods based on non-DLT, both for public and pairwise interactions.

From the process of creating a new DID results a cryptographic key pair as well as a DID Document and the DID itself. The private key associated with this key pair is extremely important and should be handled with care, as it controls the newly created identity. If someone has access to the DID master key it could easily impersonate the identity owner. Given the importance of this key the framework provides mechanisms to ensure recovery from DID master key loss. Identity owners can export their DID master keys and split it in multiple shares that can be shared with trustees. If the key is ever lost it could be reconstructed from the shares and imported back. Although if the master key becomes compromised there is no mechanism to revoke or rotate it.

For public DIDs the ability to update the corresponding DID Document is also supported since they are intended for long term use. Identity owners may change service endpoints or feel the need to rotate DID Keys, so being able to add and update keys, service endpoint and authentication methods is an essential part of identity management in the SSI ecosystem. Nonetheless pairwise DID methods may not always provide the ability to

manage keys and endpoints. While the Peer DID Method provides an updatable DID Document that relies on an append-only log to store the changes (*Deltas*), the Key DID Method uses a more simplistic approach where the the DID Document is deterministically derived from a key pair, so updates are impossible.

Besides creating and managing identifiers this module also provides functionalities for resolving DIDs into the corresponding DID Documents.

DKMS Module

The DKMS module provides an infrastructure to securely store credentials and private keys, in the form of an identity wallet. Besides storing keys and credentials the wallet can also store copies of Credentials Schemas and Revocations Registers, in the case of the identity owner being an issuer. Even though these documents are published on decentralized networks to be available to verifiers during the credential verification process, the wallet also stores a copy of the latest version of the document so issuers can check the current state of the Schema or Registry.

Information stored in the wallet is kept encrypted to protect the identity owner against potential breaches, nonetheless it can happen that the identity owner loses access to his wallet, either by losing the device that hosts the wallet or by forgetting the password to access it. So the DKMS module provides the ability to create encrypted backups of the wallet data. If an identity owner losses his wallet it can import a backup to a new instance of the wallet and remain in control of his credentials and private keys. Also the cryptographic keys used to create the backup are generated using a PBKDF, so the identity owner is the sole responsible for managing the backup passwords. This recovery mechanism is defined in the DKMS standard as offline recovery and ensures identity owners can regain access to his keys and credentials in case of wallet loss.

Besides providing a wallet for credential and key storage, the DKMS module also includes a cryptographic library for the generation of key pairs as well as for signing and verifying credentials and DID Documents. This library supports most of the signature suites and key types defined by the W3C for Linked Data documents [105].

DID Auth Module

As for the DID Auth module it provides a mechanism to authenticate entities in the SSI ecosystem, by proving ownership of a DID. For this process the framework provides a couple of different options, it implements the DID SIOP flow which was mentioned in section 3.3.2, it also provides a DID Auth flow using HTTP Signatures and another one that relies on JWTs and claims to encode the challenges. The basic idea behind the different flows is the same, the authentication requester generates a random challenge that must be signed by the identity owner to prove that it is in control of the private key corresponding to the public key presented in the DID Document, only the format of the challenges differs.

DID Comm Module

For DID Comm the framework includes an implementation of the encrypted envelopes specification. This specification defines the message structure as well as the cryptographic methods used to encrypt the DID Comm messages. These messages are used in every interaction of the SSI ecosystem, and they can also be used to implement higher lever protocol for credential exchange and request. DID Comm messages are transport agnostic so the framework does not provide any transport method, given that any data transfer protocol can be used to exchange them.

VCs Module

The credentials module provides the ability to create and verify both VCs and VPs according to the W3C specification. It also support other functionalities related to VCs such as creating and managing credential schemas and revocation registers. Credential schemas can be used to define the structure of a credentials type by specifying which attributes are included, while revocation registries provide an essential requirement of SSI that is the ability to revoke previously issued credentials.

Both schemas and revocation registries are published on decentralized networks like DID Documents so third party entities can access them during the credential verification process. They will have a cryptographic identifier that can be used to resolve them, as well as the DID of its creator and a cryptographic signature to ensure the integrity and authenticity of the document. The framework uses the IPFS and IPNS to store credential schemas and revocation registries, however other decentralized networks can also be used as long as they provide a URL to the stored data, so it can be included in the credential.

5.3 ARCHITECTURE

This section gives an overview of the framework architecture. It describes how trust is enforced in a non-DLT SSI system, how the wallet provides an interface for most identity management operations and how the different components interact to achieve the final goal of enabling SSI.

Trust in the SSI ecosystem is anchored on the DPKI, and since the DPKI is implemented using non-DLT DID Methods trust is provided by the decentralized networks themselves. Most of the decentralized networks are based on public key cryptography and identities are represented and controlled by a cryptographic key pair. This ensures that only the controller of the private key has the ability to publish data in the namaspace and trust is built around that. Furthermore every record added to the namespace needs to be signed by the identity key pair to ensure its integrity and authenticity. In the non-DLT approach trust is provided by public key cryptography which also implies that losing the key that controls the namespace would results in complete lost of control over the identifier.

To enable the DPKI functionalities such as the creation, resolution and management of DIDs the framework relies on local nodes of the IPFS, GNUnet, Freenet and Dat. These networks follow a public and permissionless approach so any entity is able to create an identity and perform one of the roles of SSI. These nodes store DID Documents and identity key pairs, but can store credential schemas and revocation registries as well. Also given that these decentralized networks do not provide complete replication of content like most DLTs, nodes also play a role in the availability of the data. While data is stored in an active node it will be available for resolution, but if the node leaves the network the data could become temporarily unavailable, if it is not replicated in other active nodes.

Besides the DPKI another major component of the framework architecture is the wallet. This component enables identity owners to store their private keys and credentials in a secure environment. But despite being a storage infrastructure the wallet is also responsible for mediating interactions with other entities in the SSI ecosystem, as well as communicating with the decentralized networks in order to create and manage DIDs. So operations such as creating and updating DID Documents, making DID Comm or DID Auth request, and issuing VCs are performed through the wallet. Moreover all of the signing and verification operations are done inside the wallet to minimize the exposure of the cryptographic keys. All of this makes the wallet a focal component in the architecture as it couples all of the main functionalities from the framework modules, enabling identity owners to manage their identities though a single component.

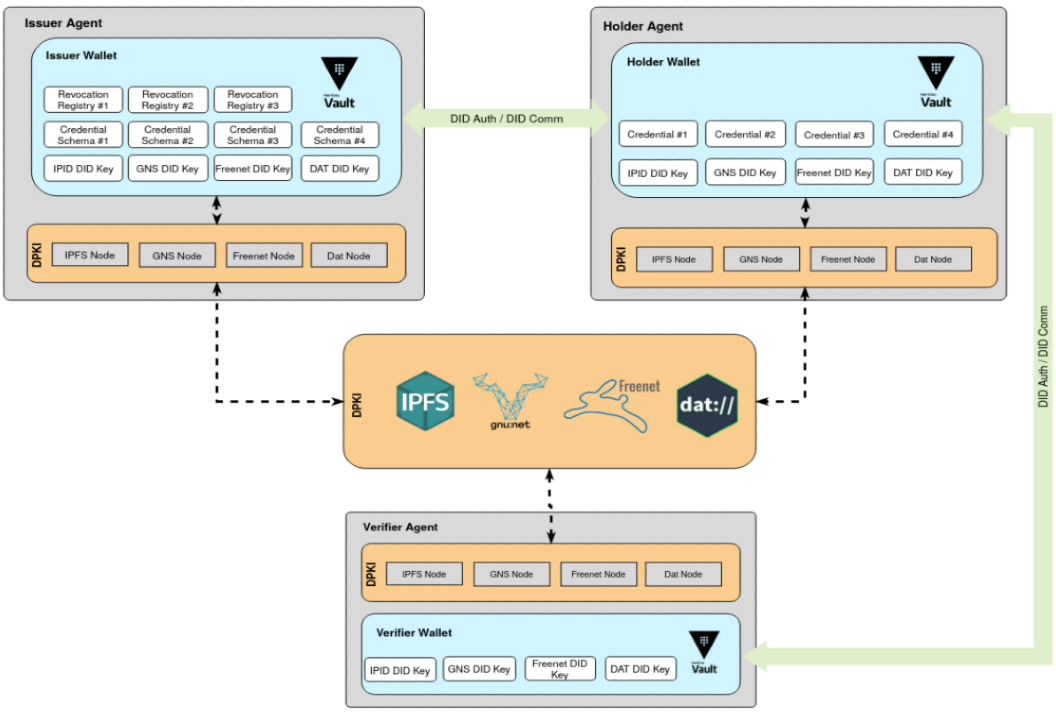


Figure 22: Framework Architecture

Also keys shown in Figure 22 represent DID sub keys, given that the wallet does not store DID master keys. This is done for security reasons to ensure that wallet compromise does not mean complete lost of control over the identifiers. If DID master keys were stored in the wallet and it became compromised the intruder would have access to all the DID master keys resulting in master key compromise and complete lost of control over the identifiers. By only storing the DID sub keys, if the wallet is compromised the identity owner can always import his master key in another node and perform an update to the DID Document rotating all of the DID sub keys compromised. DID Master keys are kept in the keystore of the DPKI nodes, however identity owners are advised to backup the DID master keys in paper format or use the provided recovery mechanism and split the key amongst trustees.

For the wallet the framework provides two different implementations, one intended for credential issuers and another for credential holders. These implementations provide different functionalities given that the two roles perform different actions in the ecosystem. While the basic operations of an identity owner, such as the creation and management of DIDs and use of communication and authentication protocols are shared by both implementations, only issuer wallets have the ability to issue credentials, create revocation registries and credential schemas while holder wallets just allow for storage of credentials as well as creation and exchange of VPs.

As for the verifier there is no need for a wallet since the role of a verifier does not require the storage of information. So the verifier relies just on the DPKI to enable DID resolution and access to credential schemas and revocation registries which are needed during the credential verification process.

5.4 PROOF OF CONCEPT

To demonstrate the functionalities provided by the framework several proof of concept applications were developed [1] [2] . These applications are web based and rely on a cloud wallet to manage the identity owner's personal data. The wallet runs on a server outside the control of the identity owner and in order to access its personal data and manage his DIDs, the identity owner must first authenticate with the wallet. This approach can take away some of the identity owner's control over its personal data, so using an edge wallet to store credential and private keys and only relying on the cloud wallet for backups and interaction with the DPKI would be a better approach, since the personal data would be kept under the control of the identity owner. However to simplify the implementation a pure cloud wallet approach was chosen.

As for the architecture of the applications it follows a client-server model where the server sets up the DPKI nodes and wallet to store the identity owners personal information. The server then provides an API so the client can interact with the wallet and manage his identity and credentials. The applications were implemented using React-JS and the Django python frameworks. With Django a Rest API was developed to serve as the application's backend and provide an interface to the SSI framework functionalities, and for the application's frontend

1 https://gitlab.devisefutures.com/devisefutures/ssi_nao-dlt
2 https://vm4.devisefutures.com/SSI_VM.tar.gz

the React-JS framework and Material-UI component library was used.

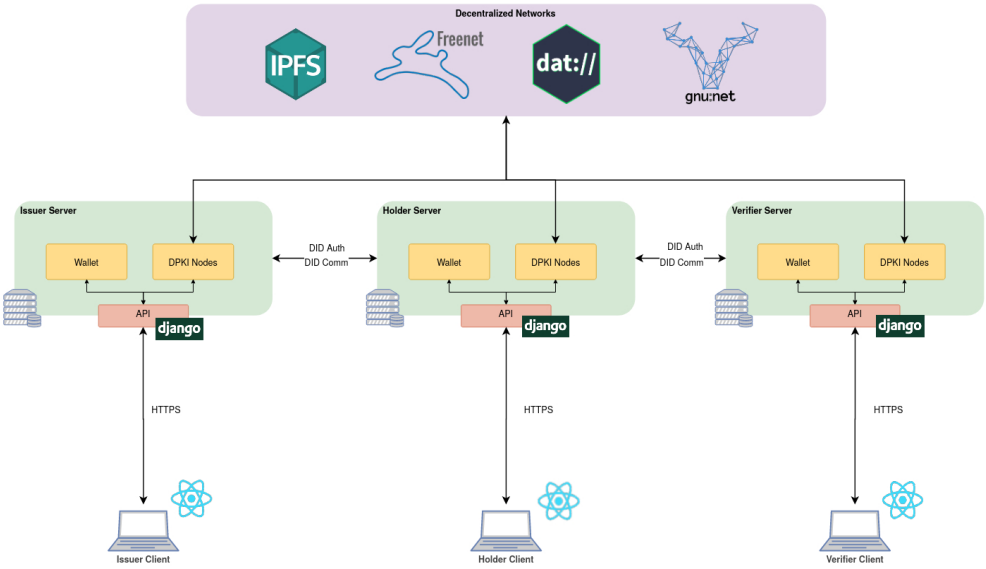


Figure 23: Application Architecture

This proof of concept includes three different agents, one for each of the SSI roles, and tries to illustrate how entities interact and exchange credentials in the SSI ecosystem. It covers a use case scenario for SSI that is the creation of a governmental identity management service. In this scenario credentials holders are able to request their eID, healthcare and passport credential amongst other from governmental entities acting as SSI issuers. Holder then store these credentials in their wallets and exchange them with service provider in order to be granted access to resources or services. In the appendix section several sequence diagrams are provided to further explain the different procedures and features of the applications with more detail. Below the different agents are presented.

5.4.1 *Issuer Agent*

In this scenario the issuer agent represent a governmental institution capable of issuing a number of different credentials related to governmental services. To request credentials from the issuer, holders must access the issuer web page and fill the forms with the required information for the desired credential. After that the credential will be automatically issued and sent to the credential holder wallet. During this process DID Auth is performed to ensure that the holder requesting the credential is in fact in control of his DID, and a DID Comm encrypted envelope is used to send the credential to the holder wallet. Figure 24 shows the issuer home page and form for the eID credential.

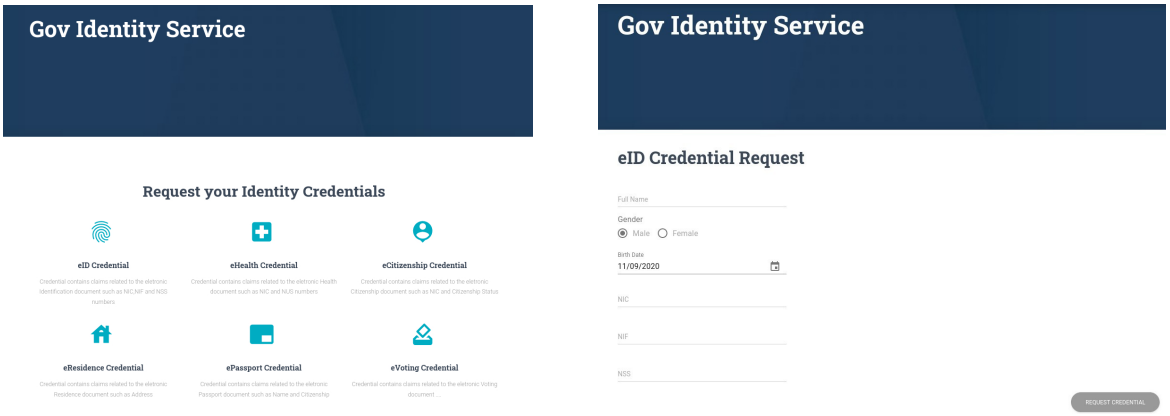


Figure 24: Issuer Credential Request Page

But before being able to issue credentials, a issuer must first set up his own wallet and create a DID to be able to interact in the SSI ecosystem. Issuers can also create credential schemas and revocation registries to enable a better management of the issued credentials. So the issuer dashboard despite providing an interface to create and manage DIDs, it also includes the ability to create credential schemas and revocation registries. These should be used by issuers to define the structure and attributes of the provided credentials and to revoke credentials when needed. Only issuers have the ability to perform credential revocation, so if a holder wants a credentials to be revoked it must contact the Issuer.

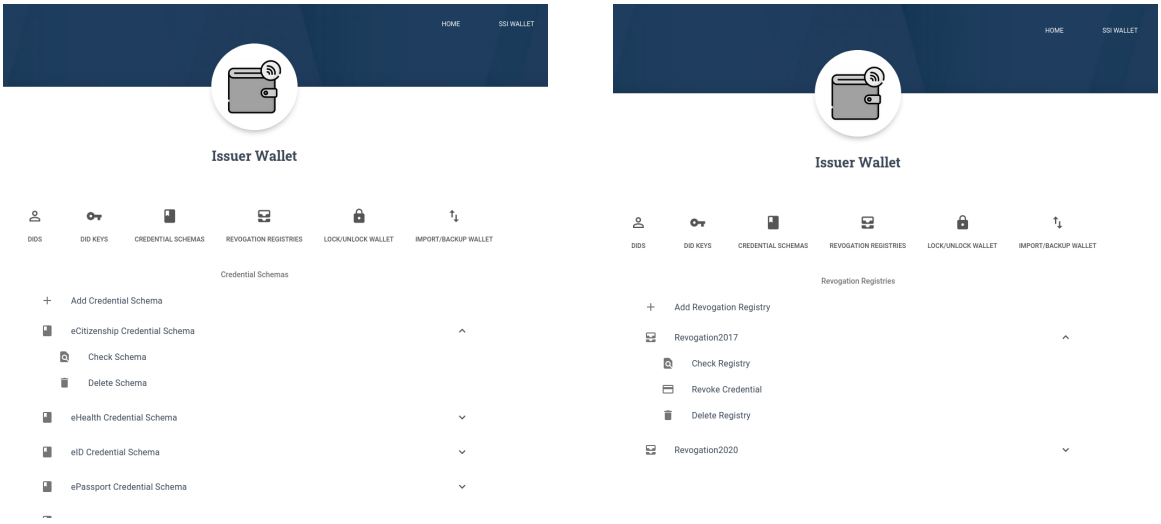


Figure 25: Issuer Dashboard

5.4.2 *Holder Agent*

Like the issuer, a holder must first create a wallet and then register a DID to start interacting and requesting credentials in the the SSI ecosystem. This is the first step to be done in the holder agent in order to get access to the Holder Dashboard. From the dashboard the holder can then create and manage his DIDs as well as store and exchange credentials. Holders can create public and pairwise DID, nonetheless to interact with credential issuers a public DID should be used.

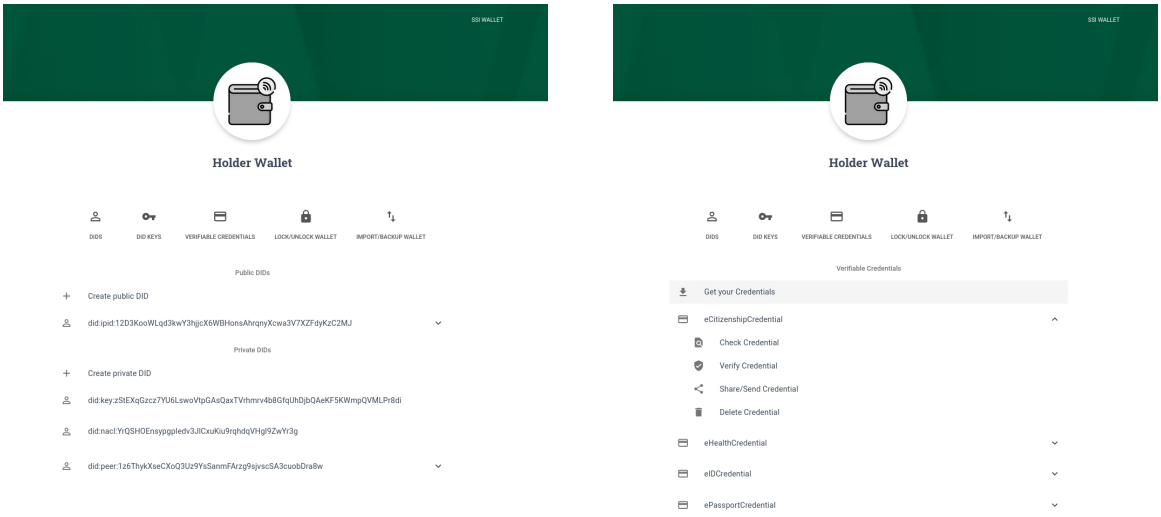


Figure 26: Holder Dashboard

From his wallet the holder can be redirected to the issuer web pages where he can requests credentials. After having credentials stored in his wallet the holder can then exchange them with service provider and verifiers in order to gain access to services or resources. During the credential exchange process a VP is created from the VC and a DID Comm encrypted envelope is used to share the presentation with the verifier. Credentials are shared via URL, and when the URL is called it will trigger an API request in the verifiers backend that verifies the credential and grant access to the holder if it is valid. The credential issuance process works in a similar way, where the issuer makes a call to the holder API to store the newly-issued credential in their wallet.

From his dashboard the holder can also check his credential which will be displayed in JSON format, and verify them to ensure they remain valid and if not, credentials can be deleted.

5.4.3 *Verifier Agent*

The verifier agent is an application for scheduling Covid-19 Vaccine appointments. To interact with this agent holders must share their healthcare credential, containing the healthcare ID number used to identify the holder in the healthcare system, in order to schedule the vaccine appointment. After the first appointment has been scheduled the holder can continue to access the verifiers' home page to check his vaccination progress, although the healthcare credential must be shared every time the holder visits this page.

This agent has no wallet or dashboard since its role does not require managing or storing any information from the credential subjects. So in this proof of concept verifiers are only able to verify presentations and resolve DIDs. Besides DID resolution, the credential verification process may also involve accessing a decentralized network in order to obtain revocation lists and credential schemas. This whole process is performed automatically by the verifies' backend after the credential has been shared with the verifier.

It can also happened that the verifier role is performed by an agent that already has another role in the ecosystem. For example during the credential issuing process an issuer can require the holder to present a valid eID credential in order to issue the healthcare or residence credential, and by doing so it is also taking the duties of a verifier. Figure 27 shows the two different cases of credential verification included in this proof of concept.

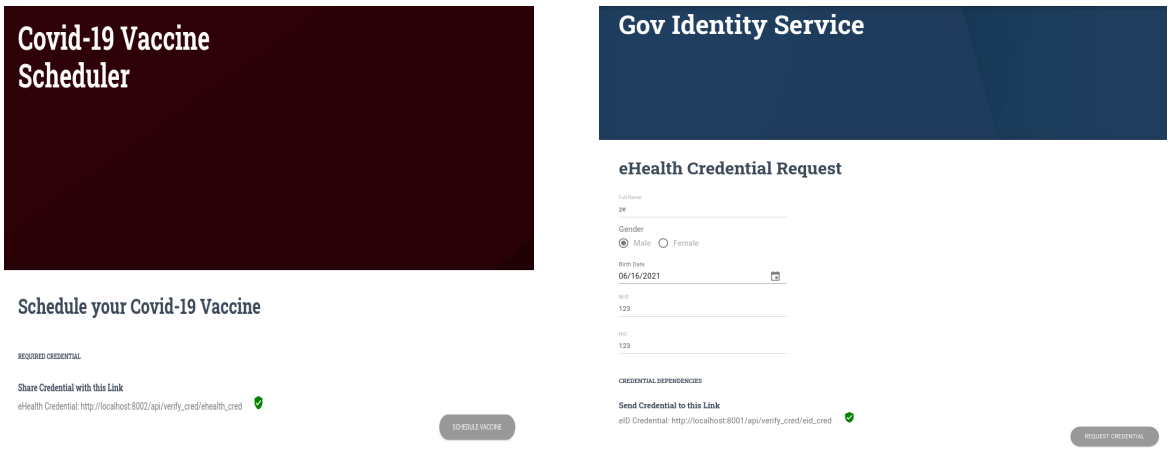


Figure 27: Credential verification

6

FINAL THOUGHTS AND FUTURE WORK

This chapter summarizes and gives some final thoughts on the use of non-DLT infrastructures to anchor trust in the SSI ecosystem. It also includes an overview of the future work and research to be done.

Final Thoughts

As mentioned throughput this document DLT infrastructures are limited in terms of performance, scalability and some of its mechanisms make compliance with data protection regulations almost impossible, so this thesis took a different approach to SSI and investigated if other decentralized infrastructures could be used to replace DLT as decentralized identity registry in SSI systems. The table bellow summarizes the main features and differences between the presented non-DLT solutions and DLTs.

	Data Encryption	Query/Traffic Encryption	Anonymity	Data Availability	Data Erasure	Identity Revocation	Name System
IPFS	✗	✓	✗	⚠	⚠	✗	✓
GNUnet	✓	✓	✓	⚠	⚠	✓	✓
Freenet	✓	✗	✓	⚠	⚠	✗	✓
DAT	✓	✓	✗	⚠	⚠	✗	✗
DLT	✓/✗	✗	✓/✗	✓	✗	✓	—

Table 1: Comparison of the different solution

When it comes to data encryption most systems enforce encryption of the data blocks when adding content to the network, with the exception of the IPFS and some DLTs. In this table data encryption refers to the encryption mechanisms provided by the network itself. Data can be encrypted before being added to the network, nonetheless most networks enforce their own encryption mechanisms to ensure that data is stored in a secure and tamper-proof manner.

Other systems like IPFS, GNUnet and DAT, also encrypt queries or traffic between the network nodes to protect the data being exchanged from malicious third parties eavesdropping the communications. Furthermore additional privacy mechanisms can be used to ensure the anonymity of the entities involved in the data exchange.

From all the system reviewed only the GNUnet and Freenet try to guarantee the anonymity of the peers interacting with the network. It is worth noting that some systems enforce query or traffic encryption, but still do not ensure the anonymity of the peer, this is due to the fact that peers often publish metadata in the DHT that often includes peer IDs and the IDs of the content it hosts. And there is the case of Freenet that does not enforce query encryption but relies on a routing algorithm to ensures the anonymity of the peers requesting data.

In the case of DLTs, most of them do not ensure the anonymity of the users given that ledger transaction often include the addresses of both sender and receiver. While these addresses are "anonymous" the fact that they are often written to a public ledger makes it possible to link all the transactions back to a specific address, taking the form of pseudo-anonymous addresses. So most DLTs only provide the pseudo-anonymous characteristic like most decentralized network also do, including the IPFS and DAT. Also permissioned DLTs which are often used in SSI do not provide anonymity since they are governed by a small group of entities that have complete control over the ledger and who interacts with it. Nonetheless there are certain DLTs, such as the Monero and ZCash ledger, that rely on ZKP to provide an identity that is non-identifiable, unreachable and untraceable. In this case the anonymity of the peer is ensured given that the ledger transactions do not reveal any information about the parties involved, only that the transaction has occurred.

One major drawback associated with the non-DLT systems is the fact that the availability of data may not be guaranteed. Unlike DLT each nodes only store a very small subset of the network information and for data to be available at least one active node must be hosting it. Nodes often leave and join the network so it can happen that the data becomes temporarily unavailable. To solve this issue third party pinning services can be used or the DKMS cloud wallet could host the DPKI nodes, ensuring the availability of data for non-DLT systems. In DLTs every node stores the entire ledger so data is always available for retrieval.

Other issues with non-DLT systems are related to data erasure. Despite most systems providing mechanism to erase data they do not guarantee erasure in the entire network, or data is erased without the knowledge or consent of the publisher due to storage constraints on the nodes or to low popularity. Still this issue seems to have an easier solution on non-DLT systems than in the DLT one, given that the latter are characterized by append-only storage. In fact the IPFS already has a proposal for a data erasure mechanism that would ensure compliance with the GDPR right to be forgotten, and similar mechanisms could be implemented in the other non-DLT systems.

Another important feature of the SSI ecosystem that most non-DLT systems lack is the ability to revoke identity keys. Given that identities are represented by cryptographic key pairs there must be a way to mark them as revoked, otherwise in case of key compromise an attacker could permanently impersonate an identity owner. Despite the relevance of this feature from the non-DLT solutions only the GNS provides a key revocation mechanism, which is based on PoW. In DLT systems, identity key revocation is often implemented using smart contracts.

Another thing worth mention is the fact that most non-DLT solution rely on name systems to enable immutable content to be dynamically referenced under a unique identifier. However the use of these name systems introduces a considerable latency overhead on the DID management and resolution operations. While DLTs seem to

be limited by their lack of scalability, most of the non-DLT systems presented are bottlenecked by the high latency of the name system operations. So relying on solutions like Dat that use a public key addressing schema instead of a name system may ensure lower latency on the identity management operations.

Future Work

This thesis tried to cover most of the SSI landscape, its standards and viable alternatives to DLT in this context, however there were some other solutions and topics that had to be left out due to time constrains or simply because they were out of the main scope of the document. So this section provides an overview of the future work and research to be done.

When it comes to the DLT alternatives, there are a few more decentralized systems that might meet the requirements necessary to implement a DPKI, though the DID standard. Such systems include the **ZeroNet** [106], **Tahoe-Lafs** [107], **Secure Scuttlebutt** [108] and **Keri** [109]. These systems follow a similar approach to the ones mentioned in section 4.3 and provide equivalent security mechanisms and privacy features. A brief analysis of these systems was done during the research for this thesis, nonetheless further research must be conducted to investigate if they can in fact be used as an alternative to DLT in SSI.

Another topic that is often related to SSI and was not given great focus throughout this thesis, is the use of ZKP to enable selectively disclose of credential attributes. ZKP are a type of proof method that enables the signer prove to verifier that they meet a certain requirement or own a piece of information without disclosing its actual value. In a SSI framework this is a must have feature given that the whole paradigm is focused on minimizing personal data exposure. The selective disclosure feature could be easily integrated into the presented framework by implementing ZKP signature schemas like the BBS+ [110] and CL signatures [40]

7

CONCLUSION

This thesis covers the decentralized identity manage paradigm that is SSI. It starts by giving an insight over the evolution of the digital identity paradigms in the last few years. This showed that users have an increasing desire to have more control over how their personal information is handled and shared with third parties, which ultimately lead to SSI. Previous centralized and federated models have clearly failed to protect users' personal information from compromise and mishandle by the identity providers, so to avoid the same shortcomings SSI moves away from the centralized approach and implements a completely decentralized infrastructure for identity management.

To better understand this paradigm and the SSI ecosystem the founding principles and architectural components were break down, and the different standards were analysed. With that we could see that SSI implements a DPKI through the DID standard, to enable identity owners to fully control their identities. Furthermore personal data, that is presented in the form of VCs, and cryptographic keys are kept in the identity owner's domain, on an encrypted wallet. When the different entities in the ecosystem want to interact, request or share credentials they use a set of protocols to securely communicate and authenticate which are also built on top of DID standard.

SSI often relies on DLTs to ensure its decentralization and provide the source of trust for the whole ecosystem, however there are a number of issues associated with the use of such technology for identity management. These issues range from the lack of scalability and performance to support an identity management system that generates thousands of records for each user, to compliance with data protection regulations such as the GDPR. This thesis focused on a non-DLT approach to SSI to investigate if some of the these limitations can be overcomed by other decentralized infrastructures. To do so it explored some alternative solution, which were mostly decentralized peer-to-peer networks and name systems, and provided an implementation of the DID standard on top of these systems to demonstrate that they can in fact be used as an alternative to DLT in the context of SSI.

These non-DLT DID methods were then integrated into a larger framework that couples all the main functionalities needed to implement the different SSI agents. This framework is also presented and reviewed in detail, describing its architecture and the functionalities provided by each of the framework modules. It is essentially a lightweight SSI framework, that relies on two major architectural components, the identity wallet used to store cryptographic keys and personal information, and the network nodes used to manage DID related operations.

The framework also provides five main modules, each one representing a SSI standards, that lay-out the features needed to implement SSI systems and agents. To further showcase the framework's functionalities a proof of concept applications was developed for each of the different SSI agents. In these applications identity owners can create and manage digital identities using DIDs, and interact with the other agents in the ecosystem to request,issue or verify credentials.

BIBLIOGRAPHY

[1] Decentralized Identifiers (DIDs) v1.0, 2019. URL https://w3c.github.io/did-core/.

[2] Christopher Allen. The Path to Self-Sovereign Identity. In *Life With Alacrity*, 2016. URL https://www.lifewithalacrity.com/2016/04/the-path-to-self-soverereign-identity.html.

[3] European Union. General Data Protection Regulation (GDPR). URL https://gdpr-info.eu/.

[4] Personal Data Protection Act Overview. URL https://www.pdpc.gov.sg/Overview-of-PDPA/The-Legislation/Personal-Data-Protection-Act.

[5] Andrew Tobin and Drummond Reed. The Inevitable Rise of Self-Sovereign Identity. *The Sovrin Foundation*, 2017. URL https://sovrin.org/wp-content/uploads/2017/06/The-Inevitable-Rise-of-Self-Sovereign-Identity.pdf.

[6] Kim Cameron. The Laws of Identity. *Microsoft Corp*, 2005. URL https://www.identityblog.com/stories/2005/05/13/TheLawsOfIdentity.pdf.

[7] Quinten Stokkink and Johan Pouwelse. Deployment of a Blockchain-Based Self-Sovereign Identity. *Proceeding 2018 IEEE International Conference on Internet of Things and IEEE Green Computing and Communications*, pages 1336–1342, 2018.

[8] Alan Priddy and Kalman C. Toth. Self-Sovereign Digital Identity: A Paradigm Shift for Identity. *IEEE Security and Privacy*, pages 17–27, 2019.

[9] Verifiable Credentials Lifecycle 1.0, . URL https://w3c-ccg.github.io/vc-lifecycle/.

[10] World Wide Web Consortium (W3C). Credentials Community Group. URL https://www.w3.org/community/credentials/.

[11] State of SSI - Google Slides, . URL https://docs.google.com/presentation/d/1HbydOI0w-T{_}FY23zCACAyHmzDq1ZvyG2tklpPSm6OQQ/edit{#}slide=id.g5467e21707{_}0{_}425.

[12] Decentralized Identifier Resolution (DID Resolution) v0.2, 2019. URL https://w3c-ccg.github.io/did-resolution/.

[13] Universal Resolver. URL https://uniresolver.io/.

[14] Christopher Allen, Arthur Brock, Vitalik Buterin, Jon Callas, Duke Dorje, Christian Lundkvist, Pavel Kravchenko, Jude Nelson, Drummond Reed, Markus Sabadello, Greg Slepak, Noah Thorp, and Harlan T Wood. Decentralized Public Key Infrastructure. *Rebooting the Web of Trust*, 2015. URL https://raw.githubusercontent.com/WebOfTrustInfo/rwot1-sf/master/final-documents/dpki.pdf.

[15] Drummond Reed. DKMS Requirements Summary. 2017. URL https://raw.githubusercontent.com/hyperledger/indy-hipe/master/design/dkms/pdf/DKMS%20Requirements%20Report%20-%2030%20June%202017.pdf.

[16] Drummond Reed, Jason Law, Daniel Hardman, and Mike Lodder. Aries RFCS - DKMS Design and Architecture v4, 2019. URL https://github.com/hyperledger/aries-rfcs/blob/master/concepts/0051-dkms/dkms-v4.md.

[17] Adi Shamir. How to Share a Secret. *Communications of the ACM*, pages 612–613, 1979. URL http://web.mit.edu/6.857/OldStuff/Fall03/ref/Shamir-HowToShareASecret.pdf.

[18] Daniel Buchner and Kirk Olynyk. Fuzzy Encryption for Secret Recovery. Decentralized Identity Foundation, 2020. URL https://raw.githubusercontent.com/decentralized-identity/fuzzy-encryption/master/fuzzy-encryption-construction.pdf.

[19] Nelly Fazio and Antonio Nicolosi. Cryptographic Accumulators: Definitions, constructions and applications. 2002. URL https://cs.nyu.edu/~fazio/research/publications/accumulators.pdf.

[20] Amy Guy, David Lamers, Tobias Looker, Manu Sporny, Dmitri Zagidulin, Daniel Bluhm, and Kim Hamilton Dufy. Encrypted Data Vaults 0.1, 2020. URL https://digitalbazaar.github.io/encrypted-data-vaults/.

[21] Markus Sabadello, Kyle Den Hartog, Christian Lundkvist, Cedric Franz, Alberto Elias, Andrew Hughes, John Jordan, and Dmitri Zagidulin. Introduction to DID Auth. *Rebooting the Web of Trust*, pages 1–31, 2018. URL https://github.com/WebOfTrustInfo/rwot6-santabarbara/blob/master/final-documents/did-auth.md.

[22] Zoltán András Lux, Dirk Thatmann, Sebastian Zickau, and Felix Beierle. Distributed-Ledger-based Authentication with Decentralized Identifiers and Verifiable Credentials. *2020 2nd Conference on Blockchain Research Applications for Innovative Networks and Services (BRAINS)*, pages 71–78, 2020.

[23] OpenID Connect | OpenID, . URL https://openid.net/connect/.

[24] Final: OpenID Connect Core 1.0 incorporating errata set 1, . URL https://openid.net/specs/openid-connect-core-1_0.html.

[25] Ivan Basart, Egido Casati, Michael B Jones, Andrés Junge, David Stark, Oliver Terbu, and Dmitri Zagidulin. Using OpenID Connect Self-Issued to Achieve DID Auth. Rebooting the Web of Trust VIII, 2019. URL https://nbviewer.jupyter.org/github/WebOfTrustInfo/rwot8-barcelona/blob/master/final-documents/did-auth-oidc.pdf.

[26] Self-Issued OpenID Connect Provider DID Profile v0.1. Decentralized Identity Foundation. URL https://identity.foundation/did-siop/.

[27] IEEE Standard for Biometric Open Protocol. *IEEE Std 2410-2019*, pages 1–25, 2019.

[28] Luis Bathen, German H Flores, Gabor Madl, Divyesh Jadav, Andreas Arvanitis, Krishna Santhanam, Connie Zeng, and Alan Gordon. SelfIs: Self-sovereign Biometric IDs. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 2847–2856, 2019.

[29] J S Hammudoglu, J Sparreboom, J I Rauhamaa, J K Faber, L C Guerchi, I P Samiotis, S P Rao, and J A Pouwelse. Portable Trust: biometric-based authentication and blockchain storage for self-sovereign identity systems. 2017. URL https://arxiv.org/pdf/1706.03744.pdf.

[30] Asem Othman and John Callahan. The Horcrux Protocol: A Method for Decentralized Biometric-based Self-sovereign Identity. *International Joint Conference on Neural Networks*, 2018. URL https://arxiv.org/pdf/1711.07127.pdf.

[31] Hyperledger Aries – Hyperledger. URL https://www.hyperledger.org/use/aries.

[32] Aries-RFCs - Features. Hyperledger, . URL https://github.com/hyperledger/aries-rfcs/tree/master/features.

[33] Aries-RFCs Concepts. Hyperledger, . URL https://github.com/hyperledger/aries-rfcs/tree/master/concepts.

[34] DIF - DID Communication Working Group, . URL https://identity.foundation/working-groups/did-comm.html.

[35] DIDComm Messaging Specification, . URL https://identity.foundation/didcomm-messaging/spec/.

[36] DIDComm Messaging - Implementers Guide, . URL https://identity.foundation/didcomm-messaging/guide/.

[37] Verifiable Credentials Data Model 1.0, . URL https://www.w3.org/TR/vc-data-model/.

[38] Verifiable Credentials Implementation Guidelines 1.0, . URL https://www.w3.org/TR/vc-imp-guide/.

[39] Verifiable Credentials Use Cases, . URL https://www.w3.org/TR/vc-use-cases/.

[40] Jan Camenisch and Anna Lysyanskaya. A Signature Scheme with Efficient Protocols. *Security in Communication Networks, Third International Conference*, pages 268–289, 2003.

[41] Suvarna K Kadam. Review of Distributed Ledgers: The technological Advances behind cryptocurrency. *Conference: International Conference Advances in Computer Technology and Management (ICACTM)*, 2018.

[42] Paul Dunphy and Fabien A P Petitcolas. A First Look at Identity Management Schemes on the Blockchain. *IEEE Security and Privacy*, pages 20–29, 2018. URL https://arxiv.org/pdf/1801.03294.pdf.

[43] Dirk van Bokkem, Rico Hageman, Gijs Koning, Luat Nguyen, and Naqib Zarin. Self-Sovereign Identity Solutions : The Necessity of Blockchain Technology. pages 1–8, 2019. URL https://arxiv.org/pdf/1904.12816.pdf.

[44] Sinică Alboaie and Doina Cosovan. Private Data System Enabling Self-Sovereign Storage Managed by Executable Choreographies. *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 83–98, 2017. URL https://arxiv.org/pdf/1708.09332.pdf.

[45] Martin Schanzenbach, Georg Bramm, and Julian Schutte. reclaimID: Secure, Self-Sovereign Identities Using Name Systems and Attribute-Based Encryption. *17th IEEE International Conference on Trust*, pages 946–957, 2018.

[46] GNU Name System. URL https://gnunet.org/en/gns.html.

[47] Christian Grothoff, Martin Schanzenbach, Annett Laube, Emmanuel Benoist, and Pascal Mainini. Decentralized Authentication for Self-Sovereign Identities using Name Systems (DASEIN). 2018.

[48] Gergely Alpár, Fabian Van Den Broek, Brinda Hampiholi, Bart Jacobs, Wouter Lueks, and Sietse Ringers. IRMA: practical, decentralized and privacy-friendly identity management using smartphones. In *10th Workshop on Hot Topics in Privacy Enhancing Technologies*, 2017. URL https://petsymposium.org/2017/papers/hotpets/irma-hotpets.pdf.

[49] Security Team, Research Almaden, and TJ Watson. Specification of the Identity Mixer Cryptographic Library. 2010. URL https://dominoweb.draco.res.ibm.com/reports/rz3730_revised.pdf.

[50] Sovrin. Home - Sovrin. URL https://sovrin.org/.

[51] Jelle C Nauta and Rieks Joosten. Self-Sovereign Identity : A Comparison of IRMA and Sovrin. 2019.

[52] Gregory Linklater, Alan Herbert, Christian Smith, and Barry Irwin. Toward Distributed Key Management for Offline Authentication : Self-Sovereign Identity using Intermediate Certificates. *ACM International Conference Proceeding Series*, pages 10–19, 2018.

[53] uPort - Tools for Decentralized Identity and Trusted Data. URL https://www.uport.me/.

[54] Paul Dunphy, Luke Garratt, and Fabien Petitcolas. Decentralizing Digital Identity: Open Challenges for Distributed Ledgers. *3rd IEEE European Symposium on Security and Privacy Workshops*, pages 75–78, 2018.

[55] Why We Don't Use Blockchain - The Identiq Blog. URL https://www.identiq.com/blog/why-we-dont-use-blockchain.

[56] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, Roger Wattenhofer, and Cornell Tech. On Scaling Decentralized Blockchains. *Proc. 3rd Workshop on Bitcoin and Blockchain Research*, pages 106–125, 2016. URL http://fc16.ifca.ai/bitcoin/papers/CDE+16.pdf.

[57] Markus Schäffer, Monika di Angelo, and Gernot Salzer. Performance and Scalability of Private Ethereum Blockchains. In *Lecture Notes in Business Information Processing*, pages 103–118. Springer Verlag, 2019.

[58] Funding - Veres One. URL https://veres.one/network/funding/.

[59] Write to the Sovrin Public Ledger - Sovrin. URL https://sovrin.org/issue-credentials/.

[60] Galia Kondova and Jörn Erbguth. Self-Sovereign Identity on Public Blockchains and the GDPR. *ACM Symposium on Applied Computing*, pages 342–345, 2020.

[61] Alexandra Giannopoulou. Data protection compliance challenges for self-sovereign identity. *Advances in Intelligent Systems and Computing*, pages 91–100, 2020.

[62] Forget erasure: why blockchain is really incompatible with the GDPR | Medium. URL https://medium.com/berkman-klein-center/forget-erasure-why-blockchain-is-really-incompatible-with-the-gdpr-9f60374e90f3.

[63] Klaus Wehrle, Stefan Götz, and Simon Rieche. Distributed hash tables. *Peer-to-Peer Systems and Applications*, pages 79–93, 2005.

[64] IPNS – IPFS Documentation. URL https://docs.ipfs.io/concepts/ipns/.

[65] An Introduction to IPFS | Infura Blog. URL https://blog.infura.io/an-introduction-to-ipfs/.

[66] IPID DID Method. URL https://did-ipid.github.io/ipid-did-method/.

[67] GNUnet, . URL https://gnunet.org/.

[68] Krista Bennett, Tiberius Stef, Christian Grothoff, Tzvetan Horozov, and Ioana Patrascu. The GNet Whitepaper. Technical report, Purdue University, 2002. URL https://git.gnunet.org/bibliography.git/plain/docs/main.pdf.

[69] K Bennett, C Grothoff, Tzvetan Horozov, and JT Lindgren. An Encoding for Censorship-Resistant Sharing. pages 1–21, 2003. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.2.800.

[70] Krista Bennett, Christian Grothoff, Tzvetan Horozov, Ioana Patrascu, and Tiberiu Stef. GNUnet - A truly anonymous networking infrastructure. *Privacy Enhancing Technologies Workshop (PET)*, 2002. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.16.1301.

[71] Krista Bennett and Christian Grothoff. GAP - practical anonymous networking. *Privacy Enhancing Technologies Third International Workshop PET*, pages 141–160, 2003. URL https://grothoff.org/christian/aff.pdf.

[72] GNU Name System, . URL https://gnunet.org/en/gns.html.

[73] Matthias Wachs, Martin Schanzenbach, and Christian Grothoff. A Censorship-Resistant, Privacy-Enhancing and Fully Decentralized Name System. *13th International Conference on Cryptology and Network Security*, pages 127–142, 2014. URL https://grothoff.org/christian/gns2014wachs.pdf.

[74] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Designing Privacy Enhancing Technologies. Lecture Notes in Computer Science*, pages 46–66, 2001.

[75] Ian Clarke, Scott G Miller, Theodore W Hong, Oskar Sandberg, and Brandon Wiley. Protecting Free Expression Online with Freenet. *IEEE Internet Computing*, pages 40–49, 2002. URL https://freenetproject.org/papers/freenet-ieee.pdf.

[76] Maxwell Ogden. Dat - Distributed Dataset Synchronization And Versioning. 2017.

[77] How Dat Works. URL https://datprotocol.github.io/how-dat-works/.

[78] Jolocom DID Method Specification | GitHub. URL https://github.com/jolocom/jolo-did-method/blob/master/jolocom-did-method-specification.md.

[79] Element DID Method Specification | GitHub. URL https://github.com/decentralized-identity/element/blob/master/docs/did-method-spec/spec.md.

[80] Peer DID Method Specification | Github. URL https://identity.foundation/peer-did-method-spec/.

[81] Key DID Method Specification | Github, . URL https://w3c-ccg.github.io/did-method-key/.

[82] Jan Camenisch and Els Van Herreweghen. Design and Implementation of the idemix Anonymous Credential System. *ACM Conference on Computer and Communications Security*, pages 21–30, 2002.

[83] IRMA - Privacy by Design Foundation. URL https://privacybydesign.foundation/irma-explanation/.

[84] Foteini Baldimtsi, Jan Camenisch, Maria Dubovitskaya, Anna Lysyanskaya, Leonid Reyzin, Kai Samelin, and Sophia Yakoubov. Accumulators with Applications to Anonymity-Preserving Revocation. In *2nd IEEE European Symposium on Security and Privacy*, pages 301–315, 2017.

[85] re:claimID - Technology. URL https://reclaim.gnunet.org/tech/.

[86] Nathan S. Evans and Christian Grothoff. R5N: Randomized Recursive Routing for Restricted-Route Networks. *2011 5th International Conference on Network and System Security*, pages 316–321, 2011.

[87] IDM Project Management Repository | Github, . URL https://github.com/ipfs-shipyard/pm-idm.

[88] IDM Concept | Github, . URL https://github.com/ipfs-shipyard/pm-idm/blob/master/docs/idm-concept.md.

[89] Peergos. URL https://peergos.org/.

[90] Dominik Grolimund, Luzius Meisser, Stefan Schmid, and Roger Wattenhofer. Cryptree: A Folder Tree Structure for Cryptographic File Systems. *IEEE Symposium on Reliable Distributed Systems*, pages 189–198, 2006.

[91] Colin Percival. Stronger Key Derivation via Sequential Memory-Hard Functions. pages 1–16, 2009. URL https://www.tarsnap.com/scrypt/scrypt.pdf.

[92] Deepanshu Choudhary and Rahul Bhagat. The Onion Routing - Symbiosis Institute of Computer Studies & Research. 2018.

[93] Bernd Prünster, Alexander Marsalek, and Thomas Zefferer. Total Eclipse of the Heart – Disrupting the InterPlanetary File System. 2020. URL https://arxiv.org/pdf/2011.00874.pdf.

[94] Guanyu Tian, Zhenhai Duan, Todd Baumeister, and Yingfei Dong. A Traceback Attack on Freenet. *IEEE Transactions on Dependable and Secure Computing*, pages 294–307, 2017.

[95] Todd Baumeister, Yingfei Dong, Zhenhai Duan, and Guanyu Tian. A Routing Table Insertion (RTI) Attack on Freenet. *2012 ASE International Conference on Cyber Security*, pages 8–15, 2012.

[96] Todd Baumeister. Fundamental Design Issues in Anonymous Peer-to-peer Distributed Hash Table Protocols. Master's thesis, University of Hawai'i, 2019.

[97] Mirko Zichichi, Stefano Ferretti, and Gabriele D'Angelo. On the Efficiency of Decentralized File Storage for Personal Information Management Systems. In *IEEE Symposium on Computers and Communications*, 2020.

[98] Fabien Geyer, Holger Kinkelin, Hendrik Leppelsack, Stefan Liebald, Dominik Scholz, Georg Carle, and Dominic Schupke. Performance Perspective on Private Distributed Ledger Technologies for Industrial Networks. In *Proceedings of the 2019 International Conference on Networked Systems*, 2019.

[99] Eugenia Politou, Efthimios Alepis, Constantinos Patsakis, Fran Casino, and Mamoun Alazab. Delegated content erasure in IPFS. *Future Generation Computer Systems*, pages 956–964, 2020.

[100] DID Specification Registries . . URL https://www.w3.org/TR/did-spec-registries/.

[101] Vault by HashiCorp. URL https://www.vaultproject.io/.

[102] Verifiable Credentials JSON Schema Specification, . URL https://w3c-ccg.github.io/vc-json-schemas/.

[103] Credential Status List 2017, . URL https://w3c-ccg.github.io/vc-csl2017/.

[104] Revocation List 2020, . URL https://w3c-ccg.github.io/vc-status-rl-2020/.

[105] Linked Data Cryptographic Suite Registry. URL https://w3c-ccg.github.io/ld-cryptosuite-registry/.

[106] M Pavithra, S Vasanth, R Rajmohan, and D Jayakumar. ZeroNet: An Overview. 2018. URL https://acadpubl.eu/hub/2018-119-14/articles/3/3.pdf.

[107] Zooko Wilcox-O'hearn and Brian Warner. *Tahoe-The Least-Authority Filesystem*. 2008. URL https://eprint.iacr.org/2012/524.pdf.

[108] Dominic Tarr, Erick Lavoie, Aljoscha Meyer, and Christian Tschudin. Secure ScuttleButt: An identity-centric protocol for subjective and decentralized applications. In *2019 Conference on Information-Centric Networking*, pages 1–11. ACM, 2019.

[109] Samuel Smith. *Key Event Receipt Infrastructure (KERI)*. PhD thesis, 2019. URL https://arxiv.org/pdf/1907.02143.pdf.

[110] BBS+ Signatures 2020. URL https://w3c-ccg.github.io/ldp-bbs2020/.

A

APPENDIX

A.0.1 *Agents*

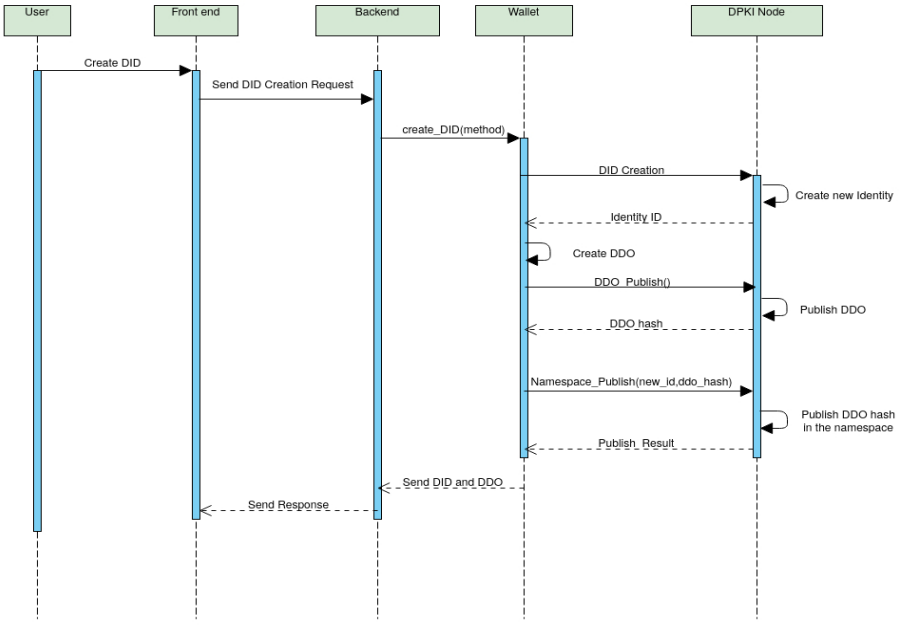


Figure 28: Sequence Diagram of DID creation

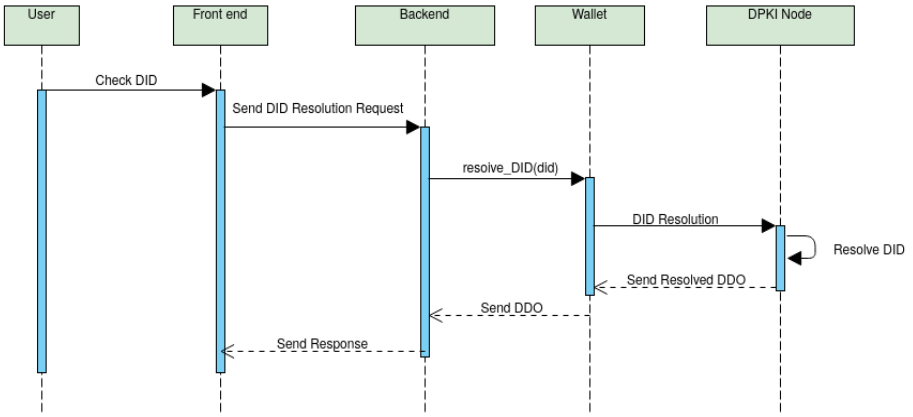


Figure 29: Sequence Diagram of DID resolution

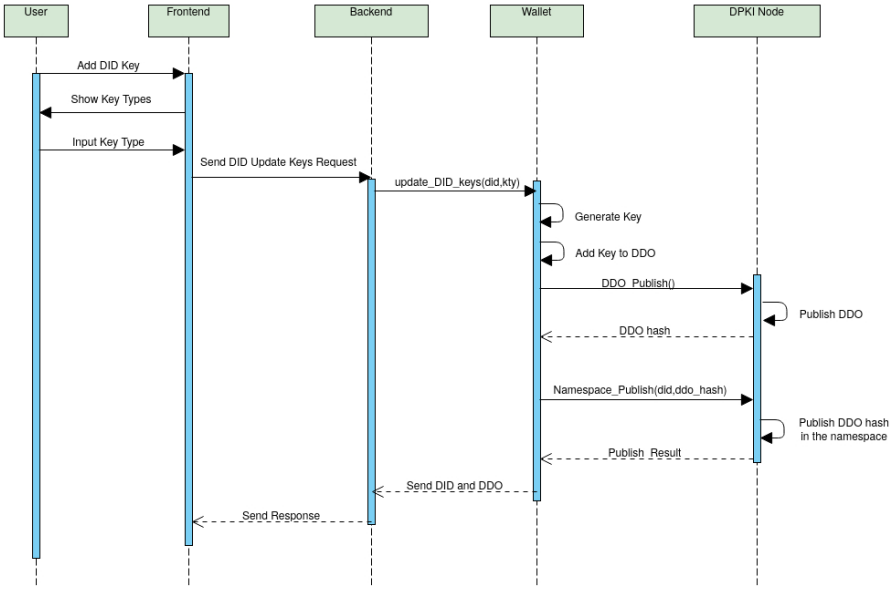


Figure 30: Sequence Diagram for adding a new key to a DID Document

Holder Agent

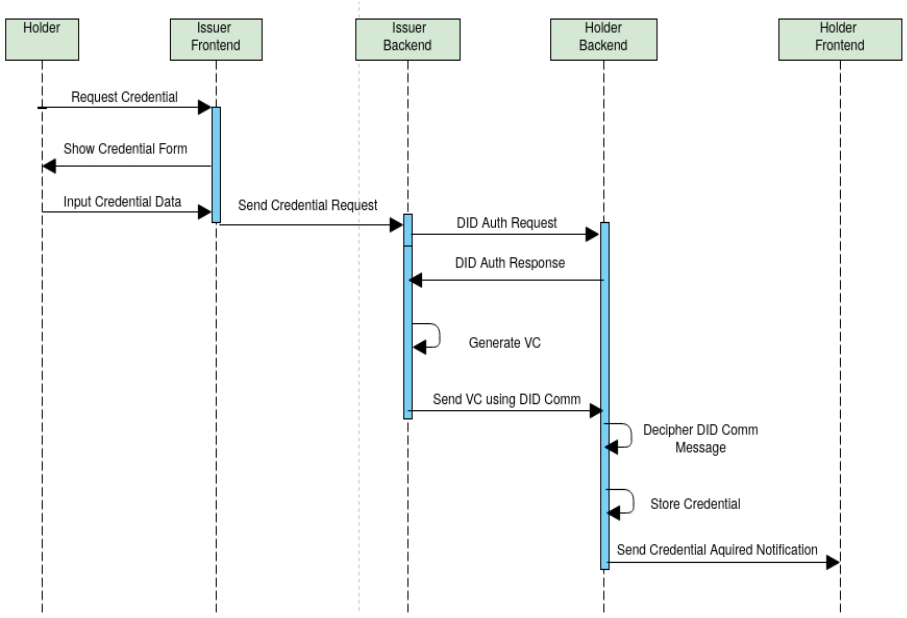


Figure 31: Sequence Diagram of a Credential Request

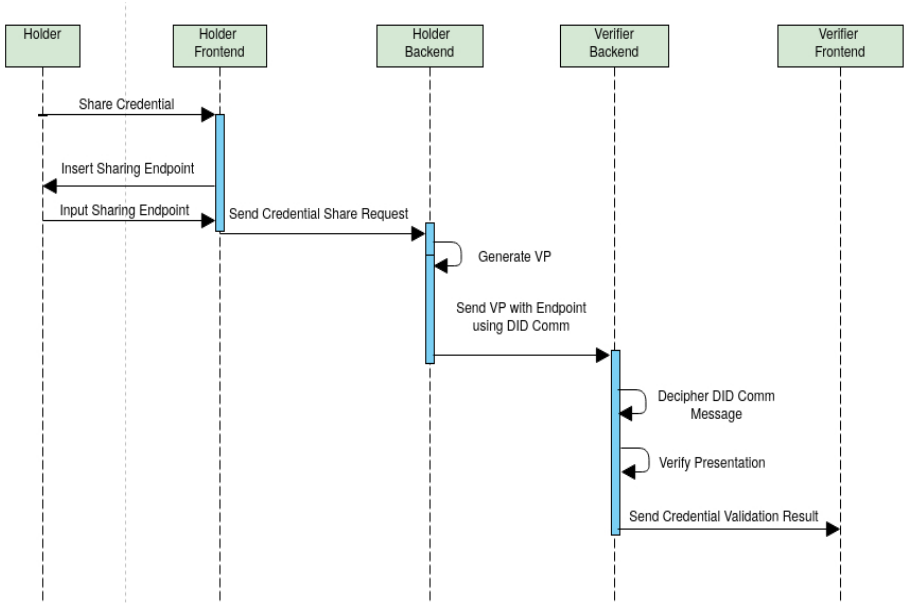


Figure 32: Sequence Diagram for Credential Sharing

Issuer Agent

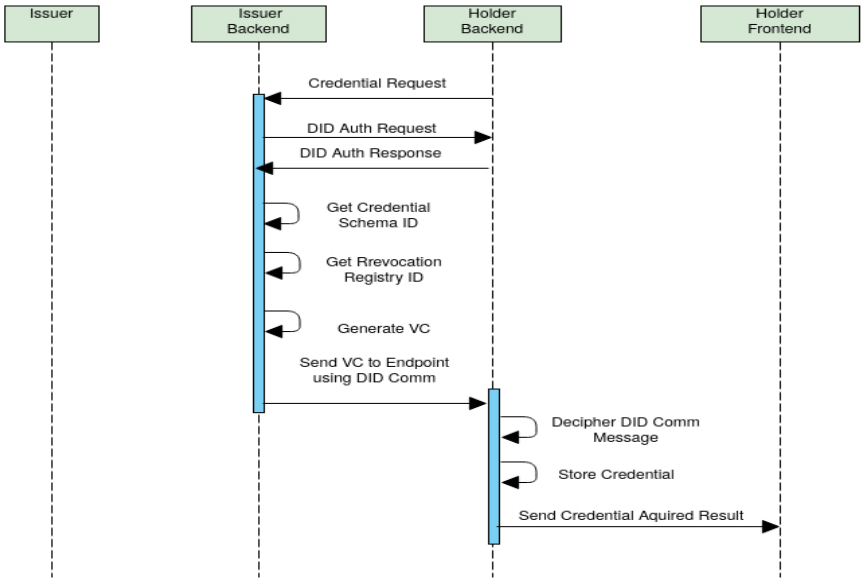


Figure 33: Sequence Diagram of Credential Issuance

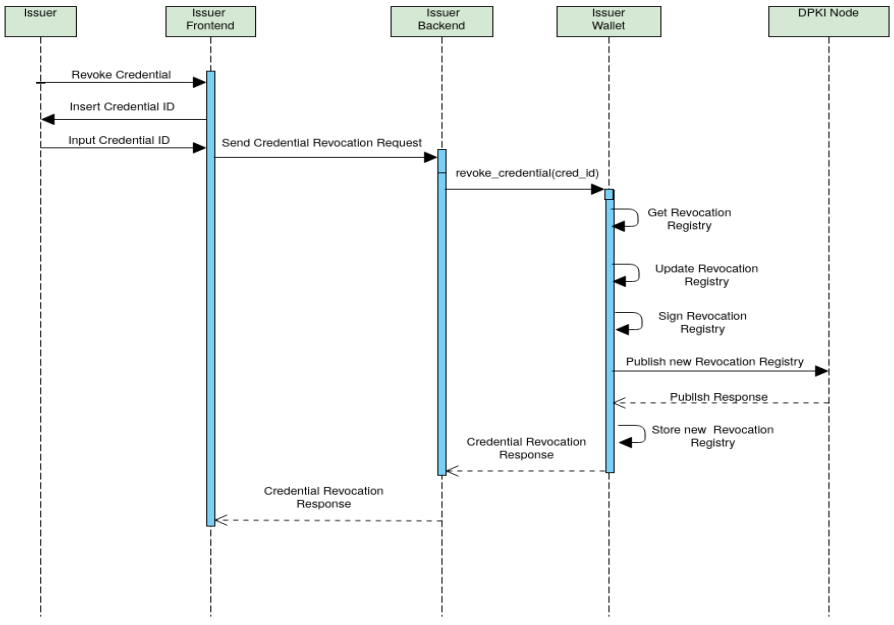


Figure 34: Sequence Diagram of Credential Revocation

Verifier Agent

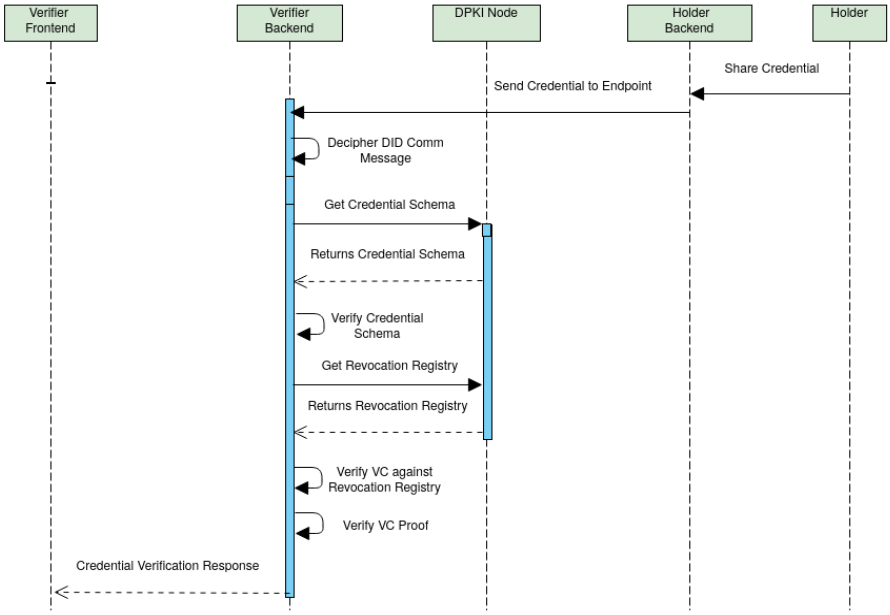


Figure 35: Sequence Diagram of Credential Verification