An Interoperable Access Control System based on Self-Sovereign Identities

Master Thesis

by

Vasil Papanchev

Degree Course: Informatics M.Sc.

Institute of Applied Informatics and Formal Description Methods (AIFB)

KIT Department of Economics and Management

Advisor:	Prof. Dr. Ralf H. Reussner
Second Advisor:	Dr. Michael Färber
Supervisor:	Christoph Braun M.Sc.
Submitted:	September 23, 2022

Contents

Abstract

The extreme growth of the World Wide Web in the last decade together with recent scandals related to theft or abusive use of personal information have left users unsatisfied with their digital identity providers and concerned about their online privacy. Self-Sovereign Identity (SSI) is a new identity management paradigm which gives back control over personal information to its rightful owner - the individual. However, adoption of SSI on the Web is complicated by the high overhead costs for the service providers due to the lacking interoperability of the various emerging SSI solutions. In this work, we propose an Access Control System based on Self-Sovereign Identities with a semantically-modelled Access Control Logic. Our system relies on the Web Access Control authorization rules used in the Solid project and extends them to additionally express requirements on Verifiable Credentials, i.e., digital credentials adhering to a standardized data model. Moreover, the system achieves interoperability across multiple DID Methods and types of Verifiable Credentials allowing for incremental extensibility of the supported SSI technologies by design. A Proof-of-Concept prototype is implemented and its performance as well as multiple system design choices are evaluated: The End-to-End latency of the authorization process takes between 2-5 seconds depending on the used DID Methods and can theoretically be further optimized to 1.5-3 seconds. Evaluating the potential interoperability achieved by the system shows that multiple DID Methods and different types of Verifiable Credentials can be supported. Lastly, multiple approaches for modelling required Verifiable Credentials are compared and the suitability of the SHACL language for describing the RDF graphs represented by the required Linked Data credentials is shown.

Introduction

The Internet and the World Wide Web have had an immense growth in the last decade. For example, in 2021 there were over 3.78 billion users of online social media [108] while the new streaming video platform TikTok averaged 167 million watched videos per minute in August 2021 [107]. Furthermore, many previously offline activities and even government services, such as shopping [96], payments [62], healthcare [37], and tax declaration[1], have evolved and are now provided as Web Services in search of optimized efficiency and better user-experience. The Covid-19 Pandemic also led to even more digital records due to forced online activities [86]. However, the attempts of mitigating the spread of the virus have caused sacrificed consumer privacy as many people were required to share health and location data [18]. Furthermore, recent privacy breaches related to theft or abusive use of personal information showcase the many downsides of the current version of the Web and its business model - companies selling their collected user data to third parties for profit [42]. For example, a publicized scandal in March 2018 revealed that the political consulting firm Cambridge Analytica reportedly collected personal information of nearly 87 million Facebook users (and of their friends without an explicit consent) to be used for political advertisement during multiple elections across the globe [56]. Moreover, even without an actual breach, merely the concern of an unprotected online privacy has been examined to lead to jeopardized offline privacy and negative emotional feelings [38].

However, users do care about the management of their online identities: a survey conducted by Innovalor [54] shows that (dutch) people often use identity management systems they do not want to use. Over 80% of the interviewed wish they had more control over their identity data and more insight over how their identity data is being used. Another technical report [79] states that people do not mind to share their identity data (as long as they see value in this sharing), however, there is an increasing need for people to retain control over their data. In a survey done by The Boston Consulting Group [93], 88% of the interviewed are concerned about how their identity data is used by corporations but only 30% have a relatively comprehensive understanding of which institutions are

[1]ELSTER, an online platform for German tax declaration - `https://www.elster.de/eportal/start`

collecting their information and for what purposes. More recently, N. C. Krämer et al. [67], state that humans have a fundamentally strong motivation to protect their online privacy, however, users of online social media often struggle to do so. Furthermore, users should be supported by technological solutions to balance their privacy with the need to self-disclose information when browsing the Internet [67].

Self-Sovereign Identity (SSI) [3] is a new identity management model or paradigm with the goal of providing decentralized digital identities while preserving the privacy of the individual. According to the SSI principles, identity owners should have autonomy over their identities and have full control over how and to whom their identity data is disclosed. The authors of [52] and [106] define SSI as a "lifetime portable digital identity that does not depend on any centralized authority and can never be taken away". On a technological level, two W3C standards represent the building blocks of an SSI ecosystem - Decentralized Identifiers (DIDs) [104], a type of Uniform Resource Identifiers (URIs) used as user identifiers that are both globally-unique and globally-resolvable, and Verifiable Credentials (VCs) [105], a standardized data model for digital credentials. The owner of a VC can use it to disclose information about themselves in a cryptographically-verifiable way. With the help of a Distributed Ledger (DLT), Decentralized Identifiers can be realized without having to depend on any type of a centralized infrastructure for registering, updating or resolving the identifiers [7].

Due to the decentralized nature of the SSI paradigm and the deliberately-flexible standardization of the underlying SSI technologies, the same problems are being solved by different interest groups and organisations resulting in heterogeneous implementations. For example, there are currently 113 officially registered DID Methods [109] and 4 broadly defined types of VCs [127]. Achieving interoperability between different SSI ecosystems is one of the major challenges for the successful integration of SSI on the Web [68]. Without convergence between the SSI technologies, functional interoperability between SSI ecosystems will not be achieved [127]. Furthermore, the complexity of the SSI technology and its high overhead make adoption of SSI by the service providers a risky strategy requiring high upfront investments [68].

As a solution to the challenges for the service providers, we advocate interoperability based on standards and semantic modelling. By analyzing multiple DID Methods, VC types and other SSI technologies, we identify and evaluate solutions for interoperability between them. Furthermore, for a simplified SSI adoption we suggest a design of a generic Access Control System (ACS). An ACS is responsible for protecting the resources of a server or application by validating whether given subjects are authorized to perform an access requested by them [77]. Because nearly all Web applications require some secure data management [51], the design and open-source implementation of an SSI-based ACS can significantly simplify (or at least improve) the process of adopting SSI technologies for the

service providers. The system design of our ACS is based on a Microservice architecture and employs multiple driver-based services which allow for incremental extensibility of the supported SSI technologies. By out-sourcing the interoperability issues to the ACS design, only the configuration of supported SSI technologies (in the form of deploying the appropriate drivers) remains for the service providers. Moreover, the access control logic of the system is defined using semantic technologies and achieves both human- and machine-understandable authorization rules that can also express required VCs in an agnostic way of their specific types.

1.1 Objective of the Work

The main objective of this work is to design, implement and evaluate an interoperable Access Control System based on Decentralized Identifiers and Verifiable Credentials. The modelling of the access control logic should be done in a user- and machine-understandable way using semantic technologies. With the results of our work we aim to motivate existing systems, e.g., the Solid Protocol, to add support for SSI-based access control, which could be a major step towards the successful integration of SSI on the Web. To achieve the defined objectives, the following steps will be undergone in the scope of the work:

- Analyse the current state of the SSI paradigm by researching challenges and benefits, involved organizations and their goals, defined standards and used technologies.
- Examine different DID Methods and types of Verifiable Credentials and analyse solutions for interoperability between them.
- Design and implement a Proof-of-Concept Access Control System capable of exchanging and verifying credentials and making authorization decisions based on required and provided types of credentials. The system should allow for defining access control logic by specifying required types of credentials in a user-understandable and semantic way.
- Evaluate key properties of the system, such as performance and interoperability with regard to different DID Methods and types of Verifiable Credentials.

1.2 Structure of the Thesis

The thesis is structured as follows. We first provide the reader with a Preliminaries chapter which gives the needed background in Public Key Cryptography (section 2.1), Distributed Ledger Technologies (section 2.2), identity models on the Web (section 2.3),

including the architecture of Self-Sovereign Identity (section 2.4), and some principles and technologies of the Semantic Web (section2.5). Then, chapter 3 summarizes related work in the realm of Self-Sovereign identity and Web Access Control. Chapter 4 presents the analysis of the current state of Self-Sovereign Identity, by listing the major challenges for the successful integration of the SSI paradigm on the Web (section 4.1), exploring multiple DID Methods (section 4.2) and types of Verifiable Credentials (section 4.3), analysing different approaches for securing a communication on the Web (section 4.4), and introducing two important protocols for exchange of Verifiable Credentials (section 4.5). After that, chapter 5 presents our design of an Interoperable Access Control System based on Self-Sovereign Identities and the implementation of its Proof-of-Concept prototype. In chapter 6, the performance of the implemented prototype and the achieved interoperability aspects are evaluated. Finally, chapter 7 summarizes the work, defines possible Future Work and concludes the thesis.

Preliminaries

This chapter introduces the reader in some topics that will be necessary for the understanding of this thesis. First, sections 2.1 and 2.2 give an introduction into some basic primitives of Public Key Cryptography and Distributed Ledger Technologies (DLTs). In section 2.3, the evolution of user identities on the Web is presented which also includes the newest model of Self-Sovereign Identity (SSI). After that, section 2.4 depicts the architecture of an SSI ecosystem and presents its two building blocks - Decentralized Identifiers 2.4.1 and Verifiable Credentials 2.4.2. Finally, section 2.5 motivates the conformance of the Linked Data principles in the Semantic Web and presents multiple Linked Data technologies that will be used throughout our work.

2.1 Public Key Cryptography

In this section, a brief introduction into *Public Key Cryptography*, also known as *Asymmetric Cryptography*, and its origin is given.

2.1.1 Public Key Revolution

In 1976, Diffie and Hellman [35] presented a novel, fundamentally different way of perceiving cryptography, which was perceived as revolutionary at the time by the cryptographic community. Previously, it was assumed that a physical in-place exchange of shared secrets used for encryption and decryption is a crucial requirement for any secure communication. Otherwise a malicious actor could always have intercepted the exchange and steal the encryption materials. However, the authors of [35] observed that not all aspects of the mathematics used in cryptography are symmetrical. Some operations are easily executed but are very difficult to be reversed. For example, it is easy to multiply large prime numbers but it is very hard to compute the decomposition of a large number to its prime factors, known as the *Prime Factorization* problem [2]. Diffie and Hellman used a similar kind of asymmetry to prove the theoretical equivalence of the following scenario: Alice

and Bob are standing at two opposite corners of a room full of other people. Alice and Bob can shout messages to each other in such a way that they come up with a shared secret. Alice and Bob's secret remains unknown to the others in the room, although they were able to hear all of the exchanged messages [61].

As a consequence of the work of these authors, as well as of the work of Ralphe Merkle, Ron Rivest, Adi Shamir, Leonard Adleman and others, three distinct public key primitives were defined - Interactive Key Exchange, Public-Key Encryption and Digital Signatures. For the understanding of this thesis, an explanation of the latter two is given. In both public key encryption and digital signatures, each user generates a pair of two keys - a private key and a public key. The private key should be kept secret while the public key is shared with the other parties with which the user wants to communicate. In the case that the communication partners are unknown, the public key can be published on a website or disseminated in another way.

2.1.2 Public Key Encryption

In *Public Key Encryption schemes* [61, chapter 12], the public key is used for encryption while the private key is used for decryption. If Alice wants to send an encrypted message to Bob, Alice should first request the public key of Bob or should look it up if Bob has already disseminated it. Alice then encrypts her message with the public key of Bob and sends the resulting cipher to Bob. Bob uses his private key to decrypt the received cipher. Defined in a more formal way a public key encryption scheme is a tuple of probabilistic, polynomial-time algorithms (Gen, Enc, Dec) with the following properties:

- The algorithm Gen (from Generate) takes a security parameter 1^n and generates a key pair (pk, sk). Each of the generated keys has a length of at least n bits.
- The algorithm Enc (from Encryption) takes a public key pk and a message m as an input and outputs a cipher $c = Enc(pk, m)$.
- The algorithm Dec (from Decryption) takes a secret key sk and a cipher c as an input and outputs a message m or a special symbol $\perp$ representing unsuccessful decryption.

For every key pair (pk, sk) generated by the $Gen(1^n)$ algorithm and every message m the following equality must be satisfied:

$$m = Dec(sk, Enc(pk, m))$$

The goal of Public Key Encryption is to achieve confidentiality of communication - only the communicating parties can decipher the messages and see their unencrypted payload.

2.1.3 Digital Signatures

While encryption ensures confidentiality, the *Digital Signature schemes* [61, chapter 13] aim to guarantee integrity and authenticity of the transmitted messages, i.e., a receiver of a message should be able to verify whether the message was indeed created by a specified sender and whether it was manipulated after that. To achieve this, the sender uses their private key to sign their message and sends both the message and the signature to the receiver. The receiver then verifies the received signature using the received message and the public key of the sender. Formally defined, a digital signature scheme is a tuple of probabilistic, polynomial-time algorithms $(Gen, Sign, Vfy)$ with the following properties:

- The algorithm Gen takes a security parameter 1^n and generates a key pair (pk, sk). Each of the generated keys has a length of at least n bits.
- The algorithm $Sign$ takes a secret key sk and a message m as an input and outputs a signature $\sigma = Sign(sk, m)$.
- The algorithm Vfy (from Verify) takes a public key pk, a message m and a signature σ as input and outputs a boolean variable indicating success or failure of the verification.

For every key pair (pk, sk) generated by the $Gen(1^n)$ algorithm and every message m the following equality must be satisfied:

$$Vfy(pk, m, Sign(sk, m)) = True$$

Beyond integrity and authenticity of messages, digital signatures can also be used to assert knowledge of a private key corresponding to a specific public key, thus proving ownership of this key pair. To achieve this, an entity playing the role of a *Verifier* requests a digital signature of a specific message (called the *challenge*) from the entity playing the role of a *Prover*. The challenge is chosen by the Verifier and must include a sufficient amount of cryptographic randomness in order to prevent Replay-Attacks. [61, chapter 13]

2.2 Distributed Ledger Technology

A Distributed Ledger, as defined by [110], is a type of a distributed database, i.e., a database where data is replicated across multiple storage devices, that allows reading published data and otherwise only appending data (deletions and modifications are disallowed). A Distributed Ledger assumes the presence of malicious nodes and uses a Consensus Mechanisms to achieve agreement on the state replications of the stored data between all (honest) nodes, i.e., between the different entities that store the data. Distributed Ledger Technology (DLT) is the common term for a technology that enables the

realization and operation of Distributed Ledgers where the distributed nodes (eventually) agree on a shared state of an immutable record of data (often represented as sequence of transactions) despite possible Byzantine Failures. A concrete realization of a Distributed Ledger is also often referred to as the "DLT". The term Byzantine Failure refers to the Byzantine Generals' Problem [69] where multiple actors (called generals) have to agree on a mutual strategy (whether to attack all together or not all) in order to avoid a catastrophic failure. However, some of the generals are unreliable. In computer science, a node experiences a Byzantine Failure if it is not behaving as expecting which might occur due to multiple reasons, e.g., internal crash of the node, failed networking communication or a malicious intend. [110]

In 2008, an unidentified person (or an organization) under the pseudonym "Satoshi Nakamoto" presented Bitcoin [83] - a peer-to-peer electronic cash system. Bitcoin is regarded as the first fully decentralized cryptocurrency to not require a centralized trusted authority [110]. It uses a so-called *Blockchain* as the underlying data structure for storing all of the approved transactions. A Blockchain is a sequence of blocks, whereas each block stores some application-specific data (in the case of Bitcoin - approved transactions) and metadata, including the hash value of the previous block in the sequence (see figure 2.1). In this way, modifying a certain block in the chain would require to update the hash values stored in all consecutive blocks after this one, which is guaranteed to be an infeasible type of an attack by the network's consensus algorithm.

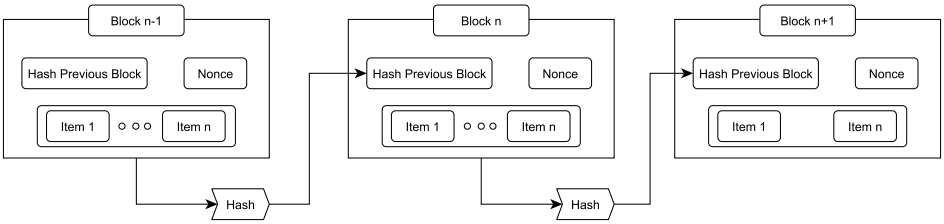


Figure 2.1: Illustration of a blockchain

Bitcoin uses a type of Proof-of-Work (PoW) (concept originally designed in [39]) consensus algorithm which requires the creators of new blocks to first solve a computationally-intensive cryptographic challenge in order for their blocks to be successfully validated by other nodes and thus appended to the chain. Other DLTs use different types of consensus algorithms, such as Proof-of-Stake (PoS), Proof-of-Elapsed-Time (PoET), Byzantine Fault Tolerance (BFT) consensus algorithms and others [99]. Due to the wide adoption of Bitcoin and the "hype" around other cryptocurrencies using the same underlying data structure, the terms DLT and Blockchain are often used interchangeably, however, the author of [110] lists three distinct primitives as the most commonly used DLT realizations:

- Blockchains [83] - DLTs using a Blockchain as the data structure for the distributed

ledger. Each block of the chain stores data represented in sets of transactions.

- BlockDAGs [71] - Block-based Direct Acyclic Graphs. The blocks within a BlockDAG can have multiple predecessors and successors forming an acyclic graph.
- TDAGs [88] - Transaction-based Directed Acyclic Graphs. The same concept as BlockDAGs, however, the graph directly consists of transactions that are directly connected to each other without being containerized in blocks.

As done by the author of [110], DLTs can be further categorized as public or private DLTs [125] and as permissionless or permissioned DLTs [126]. In public DLTs [125], any arbitrary nodes are allowed to join the DLT and participate in its maintenance procedures, e.g., validating new blocks or transactions, without having to register their identities or be approved by the network in any way. All of the information stored in public DLTs is thus accessible to everyone on the Internet. Bitcoin [83] and Ethereum [20] are examples of public DLTs. In contrast, private DLTs [125] require some authentication and verification of the participating nodes. Furthermore, the write permission is kept within a single organization. Private DLTs are often used in a specific domain where the stored information should not be revealed publicly [110]. Besides who has access to the ledger, DLTs are also categorized according to who is responsible for the synchronization of the database, i.e., for achieving a consistent state by participating in the consensus algorithm [126]. If the consensus algorithm is delegated to only a subset of all participating nodes, the DLT is categorized as a permissioned DLT. In contrast, in permissionless DLTs (such as Bitcoin [83] and Ethereum [20]), all participating nodes can validate new transactions and try to publish new ones by taking part in a consensus algorithm, such as Proof-of-Work. Depending on the diversity of the nodes responsible for managing the distributed ledger, permissioned DLTs can be more centralized or more decentralized. For example, Sovrin [92] uses the public, permissioned Hyperledger Indy Blockchain as an infrastructure for decentralized online identities. With Sovrin, only specific participants, called Stewards, are responsible for operating the nodes that maintain the Sovrin Distributed Ledger. However, Sovrin claims decentralization by having a high diversity of Stewards, each representing a different trusted organization.

In 2014, the authors of [27] presented Certcoin - a Blockchain-based decentralized alternative to Public Key Infrastructures, which enables users to look up and verify each other‘s public keys. After that, in 2016, the author of [7] first proposed using Blockchain Technology in pursuit of Self-Sovereign Identities (SSI), i.e., achieving a Decentralized Identity Management System by leveraging a decentralized, public ledger. Since then, public permissioned or permissionless DLTs and Blockchain Technology have become an integral part of most SSI realizations due to their decentralization and immutability properties.

In simple terms, the DLT acts as a replacement of the Registration Authority in classic

identity management systems. It provides the link between identification and authentication [81] by publishing bindings of identifiers to public keys. The owner of the identifier can later prove their control over the identifier (to any Verifier) using their private keys and asymmetric cryptography primitives such as digital signatures. As long as the Verifier trusts the immutability property of the DLT, they will be convinced in the trustworthiness of the identifier‘s owner.

2.3 Identities on the Web

Identity management of individuals and websites on the Web has gone though several stages, each building a new model of the online identity ecosystem and its processes. Figure 2.2 shows the four broadly defined stages of how the initially simple, centralized and isolated identities have evolved to the complex, decentralized, user-controlled and portable Self-Sovereign Identities [3].

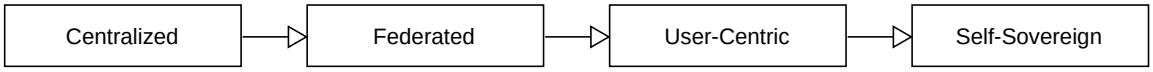


Figure 2.2: The Evolution of Identity Models on the Web

2.3.1 Centralized Identity

The earliest and still most used identity management model can be traced back to the 1990's when Sir Tim Berners-Lee founded the World Wide Web [10]. The creation of the HyperText Transfer Protocol (HTTP) [44] and the HyperText Markup Language[1] (HTML) was an important milestone in making the Web easier to understand and use. The new intuitive and user-friendly design of the Internet made it possible for the private individuals to create and post content on the Web. A new problem emerged - how could users be sure of the trustworthiness of the existing websites? The Public Key Infrastructure (PKI) [82] offered a solution - websites' identities could now be validated by verifying their certificates issued by the trustworthy Certificate Authorities (CAs). Furthermore, using Public Key Cryptography and the HTTPS protocol, these certificates could also secure the communication between websites and their clients. In this model, CAs became the central entity responsible for the management of internet identities and public keys. The drawback of this approach is that the model relies on some types of a central authority (or a hierarchy tracing back to a single root authority) to register and

[1]HTML Specification: `https://html.spec.whatwg.org/`

manage identities. Such a centralized architecture always introduces the Single Point-of-Failure drawback, as compromising any CA leads to a security breach with a high number of potential user victims. The PKIs and their alternatives are discussed in more detail in sec.4.4.

PKIs and CAs are mostly used for identification and verification of server identities. However, the growth of the Web has led to numerous use cases where some type of control over users' identities is also a necessity. The earliest identity management model for individuals is the *Isolated User Identity Model* [60], in which each service provider also plays the role of an identity provider for its users. In order to use a given service, the user creates a new identity specifically for this service, e.g., by registering a new username and password stored on the service's servers. This new identity is strictly isolated and only valid in the context of this specific service, therefore, the user must create and manage a huge number of identities - one for each service the user wants to use. This isolation does not scale with the vastly growing number of services on the Web and is one of the main drawbacks of this model [60].

Furthermore, such isolated identities in the centralized models often do not legally belong to the users, but to the centralized service providers [115]. In many cases, during identity registration, users certify that they have read and approved the terms and conditions of this service which often allow the service provider to control and even delete user identities when seen appropriate. For example, the terms and conditions of Facebook[2] state their right to terminate users' accounts:

> "*Account suspension or termination*
> *Our right to terminate for good cause remains unaffected. Good cause is specifically if a party violates obligations under these Terms, any laws, third-party rights or privacy policies, and...*"

Other notable examples for centralized identity management are any services based on the Kerberos authentication protocol [85]. It relies on a centralized authority to register and manage user identities and provides authentication services to the other entities within the domain of this service provider.

2.3.2 Federated Identity

The *Federated Identity Management* model [101] is the second stage of evolution of online identity models. It tackles the isolation drawbacks of the purely centralized approaches by defining a *federated identity domain* in which multiple service providers depend on a

[2]https://www.facebook.com/terms.php

single identity provider to authenticate users. Once a user has authenticated themselves to the identity provider, they can use all services within this federated domain, thus achieving a type of Single-Sign-On (SSO) solution. A core characteristic of this model is that all services trust the single identity provider. This makes the Federated Identity Management a suitable option mainly for use cases such as in institutional domains, where identity data is only shared between trusted entities. The most popular technology used for Federated Identity is the OASIS Security Assertion Markup Language (SAML) [90].

2.3.3 User-centric Identity

In 2003, the Augmented Social Network (ASN) [59] presented a model of a next-generation online community that integrates a persistent online identity into the architecture of the Internet. One of the core ideas of ASN is that each individual must have the right of controlling their own identity. With this, ASN laid the foundations for a new identity model that puts users instead of identity providers in the center of the identity process.

In the *User-centric Identity Management* model [60] multiple service providers can interoperate with multiple identity providers and let users select which of the supported identity providers to use. The users also have more control over what identity attributes and under which conditions to be disclosed to the service providers. The most used protocols in this model are The OAuth 2.0 Authorization Framework [50] and the OpenID Connect [98] identity layer built on top of OAuth. In OAuth, instead of the users having to authenticate themselves to the service providers, the service providers play the role of clients and are authorized by the users to access their identity attributes stored by the users' identity providers. The service providers then use the querried identity attributes to decide whether to provide their services to the users or not.

User-centric identities are successful in achieving an interoperable identity layer without requiring federated trust between service and identity providers. They also allow for some level of user privacy by requiring user consent before sharing identity data. This has been an important milestone in the development of online identities but is not the final one. The billions users of the two biggest identity providers - Facebook and Google, make them the only preferable option for service providers. Users are often given the Hobson's choice - either use one of these providers and comply to their terms and conditions, or do not use the services at all. Furthermore, the business model of these identity providers is centered around selling the collected user data to the service providers which can then use it for personalized content and advertisements [42]. Although beneficial in many situations, such a model introduces unjustified trust in the service providers and many privacy risks. Furthermore, user-centric identities are not providing true user autonomy as the identities are still controlled by the identity providers and may be deleted which then

leads to loss of identity. Although user-centric, the model still introduces centralization, Single Points-of-Failures and high privacy risks which proves that "being user-centric isn't enough"[3].

2.3.4 Self-Sovereign Identity

The next and *(currently)* final stage in the evolution of online identities is the *Self-Sovereign Identity (SSI)* model which was first defined by Christopher Allen in their seminal blog post [3] about Online Identities. To realize *Sovereignty on the Internet* means that users, organizations and even things should have true autonomy over their identities and full control over how and to whom their identity data is disclosed. To achieve this goal, new decentralized protocols for identity registration, secured communication, issuance, storage, exchange and verification of identity data must be defined and standardized. The authors of [106] and [52] define Self-Sovereign Identity as a "lifetime portable digital identity that does not depend on any centralized authority and can never be taken away". Due to the decentralized manner of this model, different organizations were expected to implement and provide Self-Sovereign Identities. To promote interoperability among them and a shared vision for their goal, Christopher Allen defined Self-Sovereign Identity in a technology-agnostic way with the following ten guiding principles [3]:

1. **Existence.** Users' existence is and must remain independent of their online identities. Self-Sovereign Identities should not try to completely represent or substitute the users. Instead, they should only provide digital access to some limited aspects of their already existing non-digital owners.

2. **Control.** Users are in control of their identities. They must be capable of referring to them, updating them, hiding and even deleting them at any time. They choose when and what identity data to disclose.

3. **Access.** Users must always be able to access all of their identity data. There should not be other entities, such as intermediaries or gatekeepers, that prevent users from accessing their data. All claims made about a user must be known to the user. This does not mean, that other entities cannot publish data about this user, but only that the user must be aware of that.

4. **Transparency.** All systems, algorithms and operations in a Self-Sovereign Identity ecosystem must be transparent. The implementations should be open-source and the governance of any supporting infrastructure, e.g., a distributed ledger, must be public so that anyone can examine their operation.

5. **Persistence.** Self-Sovereign Identities should be long-lived. They should last at least for as long as the user wishes. The lifespan of identities should be independent of any necessary rotation of cryptographic materials or updates of identity data.

6. **Portability.** Identity information and services should be transportable and not dependent on any singular third-party entity. Otherwise, the disappearing of these entities can lead to loss of identities. This principle aims to improve the independence of identities and user's control over them.

7. **Interoperability.** Self-Sovereign Identities and their services should be usable in different contexts, websites and even cross-countries. Furthermore, the usage of the identities should be done in a standardized way, so that different realizations of Self-Sovereign Identities can interoperate with each other.

8. **Consent.** Sharing of identity data must only occur after an explicit consent of the user, i.e., users select when and what information about their identities to be disclosed.

9. **Minimalization.** The Self-Sovereign Identities should allow for sharing only the minimum amount of data necessary to access a specific service. As an example, if a service provider requires its users to be of specific age, e.g. over 18 years old, the users should only assert that they comply to this rule without having to reveal any additional information such as names and the exact date of birth. Technologies such as selective disclosure, proof ranges and Zero-Knowledge Proofs can be used to achieve this.

10. **Protection.** Self-Sovereign Identities must protect user rights. As a consequence, the user rights should always have priority over the needs of the identity networks. To achieve this, identity operations should be based on censorship-resistant and force-resilient algorithms executed in a decentralized manner.

2.4 Architecture of Self-Sovereign Identity

A. Mühle et al. [81] define the SSI architecture as an *Identifier Registry Model*. In the basis of this model is the Verifiable Registry (often realized as a DLT) which substitutes the registration authority in the classic identity management systems. This allows for a decentralized and user-centric infrastructure for the management of user identifiers which on the other side make decentralized authenticated identification possible. The authors also define the *Claim Registry Model* as an extension to this architecture, in which the Verifiable Registry provides the additional functionality for the storage and management of the credentials schemes definitions and cryptographic fingerprints of issued credentials.

In Self-Sovereign Identity ecosystems, entities with independent identities connect to each other in a Peer-to-Peer manner in order to attest and validate information about themselves. Figure 2.3 shows the three main roles of SSI - Issuers, Holders and Verifiers, and the interactions between them. Every entity can play any of the three roles in different scenarios. Each entity has its own Decentralized Identifier (see sec.2.4.1) anchored on some type of a Verifiable Registry, e.g., a distributed ledger. The Issuer validates information about another entity and decides to issue a credential (see sec.2.4.2) to it. A credential contains an identifier of the Issuer, the validated information about the credential's Holder and a digital signature created with the Issuer's private key. The Holder receives the credential from the Issuer and stores it locally in their digital wallet. The Holder can then decide to disclose some information to a Verifier by creating and sending him a presentation. The presentation contains identity data from one or multiple credentials and a digital signature created by the Holder. The Verifier receives this presentation and is capable of verifying both the contained credentials using the public keys of their Issuers, and the Holder's identity by using their public key. It is important to note, that the verification occurs in a decentralized manner, i.e., the Verifier does not need to contact the Issuer in order to verify the credential issued by him. (the paragraph was summarized from the survey presented in [103])

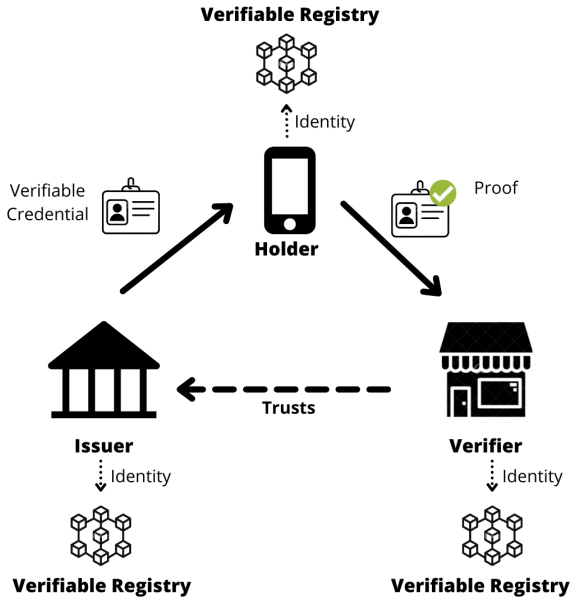


Figure 2.3: SSI Roles and Interactions

2.4.1 Decentralized Identifiers

One of the key functionalities required to achieve the goals of Self-Sovereign Identities is a decentralized registration and management of globally unique and resolvable identifiers. Previously, all persistent identifiers on the Web relied on some type of a centralized

administration either for registration (IP addresses and DNS names), or for resolving them to information about the things they identify (UUIDs [70] and URNs [97]). The *Decentralized Identifiers (DIDs)* - a W3C Proposed Recommendation for verifiable, digital identifiers [104], are a new type of Uniform Resource Identifiers (URIs) [11] that are both globally-unique and globally-resolvable without relying on any type of centralized infrastructure [106]. Additionally, DIDs also provide the ability to cryptographically verify the ownership of the identifier. A DID resolves to a *DID Document* containing public information, such as public keys and service endpoints, about the *DID Subject* - the entity identified by this DID. The DID Subject stores the corresponding private keys locally which it can use to prove control over its identifier by creating digital signatures in a challenge-response protocol. In most cases, DIDs are anchored on an underlying verifiable registry, e.g., a distributed ledger. The purpose of this registry is to provide an immutable single source of truth allowing for a decentralized registration and resolution of DIDs. [106]

It is important to mention, that the W3C recommendation only specifies some aspects of how DIDs should be implemented, e.g., general syntax, data model, core properties and required operations, such as updating and resolving DIDs. It does not specify concrete cryptographic algorithms or realizations of the verifiable registry. There are many different implementations of Decentralized Identifiers, called *DID methods* [109]. The syntax of DIDs (see fig. 2.4) uses the same pattern as the one defined in the specification of URNs. DIDs are separated in three parts - the DID scheme identifier, a DID Method Identifier and a method-specific identifier. Different DID Methods use different verifiable registries. For example, did:btcr [4] and did:ethr [118] use existing distributed ledgers, such as Bitcoin and Ethereum, while did:sov [72] and did:indy [29] implement new registries specifically designed as an underlying infrastructure for Self-Sovereign Identities. The resolution of different types of DIDs depends heavily on the used verifiable registry. There are even DID Methods that only provide local DIDs without any verifiable registry (for examples see sec. 4.2). One of the objectives of this thesis is to research and design an extensible system that is interoperable with multiple DID Methods.

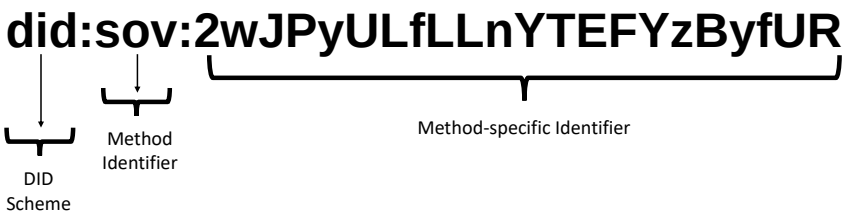


Figure 2.4: Syntax of a Decentralized Identifier

2.4.2 Verifiable Credentials

In this section, Verifiable Credentials are presented. A Verifiable Credential [105] is an official W3C Recommendation which specifies the data model for Web credentials that can be used by their owners to prove statements about themselves to Verifiers who, on the other hand, can cryptographically verify the authenticity and integrity of these statements.

People use credentials on a daily basis to assert that they are capable or authorized of doing any kinds of activities, such as entering a country, driving a car, voting, ordering alcoholic beverages, using public transportation or entering any closed-area event after proving SARS-CoV-2 vaccination. There are three roles of actors in any credential ecosystem - issuer, holder and verifier. An issuer issues a credential to a holder. The holder uses this credential to prove the attributes certified by it to a verifier. The verifier will be convinced of these attributes as long as they trusts the issuer of the credential and its validity and authenticity. Paper-form credentials are not well-suited to be used on the Web due to multiple reasons [17]. Firstly, digitally proving ownership over a paper credential is not straightforward. In most cases, it requires some biometric data contained in the credential and a way for the verifier to verify it, e.g., by interacting with the Holder on camera. Secondly, once digitized, paper credentials lose many of their anti-counterfeiting measures. Such solutions do not scale and it is clear that a new form of cryptographically-secured and machine-verifiable online credentials is needed.

In this work we differentiate between the terms Claim, Verifiable Claim, (digital) Credential, Verifiable Credential and Verifiable Presentation.

Claims and Verifiable Claims

A Claim is merely a statement about a subject that can be described as a subject-property-value relationship (see figure 2.5). Claims can be made by anyone about anything and are not inherently true or verifiable.

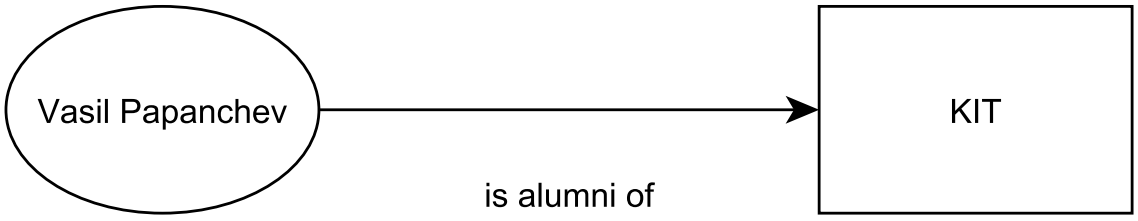


Figure 2.5: Example of a claim

A Verifiable Claim[3] is a claim that has been attested by some Issuer and this attestation

[3]Differentiation between Claims and Credentials: `https://github.com/w3c/vc-data-model/issues/112`

can be verified in a cryptographic way. Once the claim has been verified, it is up to the Verifier to decide whether to believe this claim to be true or not. This depends on whether the Verifier trusts the Issuer of the claim. A Credential is a set of one or multiple Verifiable Claims together with the metadata (identifiers of issuers, validity periods, whether the credential has been revoked, etc.) required to verify them. The definitions of Claims and Verifiable Claims have been taken from [81].

Verifiable Credentials and Verifiable Presentations

The terms Verifiable Credential and Verifiable Presentation are defined in the official W3C Recommendation for the Data Model of Verifiable Credentials [105]. A Verifiable Credential (VC) is a tamper-evident credential together with all its metadata and cryptographic proofs (see figure 2.6). The metadata of the VC contains information about the issuer, a timestamp, expiration date and time, public key used for verification, revocation mechanisms and other. The content of the cryptographic proof depends on its type. In case of a digital signature, the content may include the type of the proof, e.g., *Ed25519Signature2020*, timestamp of creation, verification method, nonce and the signature value.

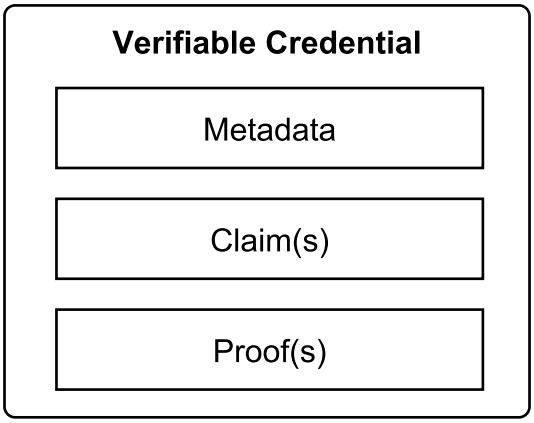


Figure 2.6: General content of a Verifiable Credential

A VC can be represented as two informational graphs - a Credential Graph and a Credential Proof Graph. The Credential Graph expresses the claims made about the subject while the Credential Proof Graph expresses the digital proof of the credential, e.g., a digital signature together with its metadata.

There are two popular mechanisms for specifying the subject of the credential, i.e., the entity about which the claims of the credential are made. The straightforward way is to include some identifier, e.g., a DID, of the subject inside the credential. Assuming that this identifier is resolvable to a public key, the Holder of the credential should only prove ownership of the corresponding private key by creating a digital signature when

sharing the credential to the Verifier. Alternatively, *Anonymous Credentials* [64] try to achieve unlinkability of credentials, i.e., credentials cannot be correlated across multiple presentations. Instead of the Holder's identifier, the Issuer embeds a commitment of a *Link Secret* that is only known to the Holder, in the credential. When presenting a credential, the Holder uses Zero-Knowledge Proofs techniques to prove ownership of the credential without revealing neither the credential itself nor the Link Secret.

Verifiable Presentation

The Verifiable Presentation defines how the Holder organizes and presents claims from one or multiple VCs to Verifiers (see figure 2.7). Firstly, a VP contains some metadata about the presentation such as type and terms of use. Then, the main content of a VP is the information from the credentials that comprise this presentation. A VP may contain one or multiple entire VCs or only a verifiable data format derived from them. In both cases, the contained information also includes the necessary proofs for its verification. Furthermore, a VP also contains some type of proof of ownership of the contained credentials. The most popular choices include a JSON Web Signature (JWS) [57] when using JSON Web Tokens (JWTs) [58], Linked Data Signatures [74] in case of Linked-Data Credentials, or Camenisch-Lysyanskaya ZKPs [22] when using Anonymous Credentials [64].

A Verifiable Presentation is represented as at least four informational graphs. The Presentation Graph refers to one or multiple Credentials or their representations, each being represented with a Credential Graph and a Credential Proof Graph. The VP also contains a Presentation Proof Graph which expresses a digital proof (usually a digital signature) together with metadata proving that the Verifiable Presentation was created by the Holder of the contained credentials.[4]

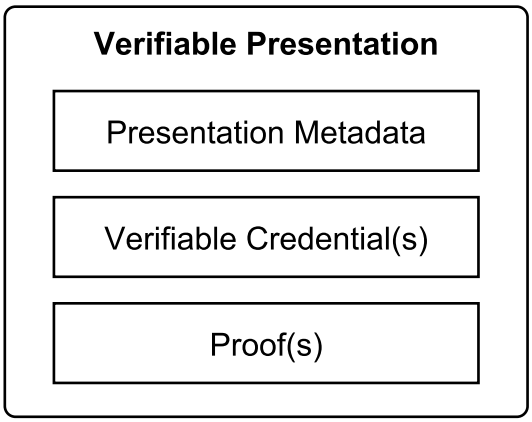


Figure 2.7: General content of a Verifiable Presentation

[4] `https://www.w3.org/TR/vc-data-model/#info-graph-vp` - Examples for VC and VP graphs

2.5 Semantic Web

The term *Semantic Web* [12] originally introduced by Sir Tim Berners-Lee refers to an extension of the World Wide Web by enriching the data on the Web with metadata that encodes its semantic meaning so that it can be processed automatically by machines. In this way, automated software could execute sophisticated tasks on the Web that require the understanding of the semantic meanings of the processed information. Furthermore, data produced by one entity can be processed outside of this context by other entities which remain sure of the semantics of the data. An example within the context of SSI would be an attribute stored in a Verifiable Credential - an Issuer creates a credential which certifies the address of a given Holder. If the credential is created according to the principles of the Semantic Web, it allows any Verifier to access the semantics of the address schema used by the Issuer, thus knowing how to parse and understand the address stored in the credential.

To realize this desired version of a Semantic Web, publishers of information on the Web should follow the Linked Data Principles [14]:

1. Use URIs as names for things.
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL)
4. Include links to other URIs. so that they can discover more things.

Following these principles allows to identify data in a consistent way and to infer its meaning by following the included hyperlinks to additional documents containing metadata, such as definitions of how to reason about the data, and even more hyperlinks to other linked information. In this way, a type of *Web of Data* [16] is achieved where data, its meaning and its relation to other data are described in a standard format (RDF) that can be used and understood cross-websites.

Due to the distributed nature of this Web of Data, it is possible and even likely that different systems will use different identifiers for the same things or, even worse, use the same identifier for different things. To solve this issue which is similar to the Tower of Babel problem, the concept of Ontologies is used. The author of [102] defines an *Ontology* (in the context of Information Science) as a "dictionary of terms formulated in a canonical syntax and with commonly accepted definitions designed to yield a lexical or taxonomical framework for knowledge-representation which can be shared by different information systems communities". In this way, different information systems, e.g., websites, can use

the same ontologies which formally define the terms and relations among them used by these information systems. Users of the websites can access the referenced ontologies and infer the intended meaning of the posted information [12]. Table 2.1 shows some of the most commonly used ontologies in the Linked Data community.

Ontology	URI	Description
Friend of a Friend (FOAF)	`http://xmlns.com/foaf/0.1/`	Describing persons, their activities and their relations to other people and objects.
DCMI Metadata Terms (dcterms)	`http://purl.org/dc/terms/`	Metadata terms used to describe a wide range of resources
Schema.org	`http://schema.org/`	Structured-Data markups used for standardizing HTML. Search engines rely on schema.org to improve the display of search results
VCard	`http://www.w3.org/2006/vcard/ns`	Description of VCards (electronic business cards) including fields such as organization, email and IMPP

Table 2.1: Popular Ontologies

2.5.1 Resource Description Framework

The Resource Description Framework (RDF) [100] is a framework for describing information about resources and the relationships between them. A resource is an abstract term and can thus be anything - physical things, online documents, people, organizations and even abstract concepts. RDF is used to encode the semantic meaning of information in a standardized way so that it can be processed automatically by different applications without loss of meaning.

RDF defines a simple data model used for expressing statements about resources. Every RDF statement has the following structure:

<subject> **<predicate>** **<object>**

The statement expresses a *directional relationship* between a **subject** and an **object**. The nature of this relationship is defined by the **predicate**. RDF statements are called

triples because they always consist of three elements. The term "RDF Term" refers to any specific piece of data that appears in the subject or object position of a triple. Note that, the same RDF Term might (and usually does) play the role of a subject in some triples and of the object in others. Because of the directional relationships, a set of RDF triples can always be represented as a direct graph. Subjects and objects are represented as graph nodes while predicates are presented as directed arcs. Figure 2.8 shows the resulting graph for the following set of RDF statements:

<Vasil Papanchev>	<is a>	<Student>
<Vasil Papanchev>	<studies at>	<KIT>
<Vasil Papanchev>	<is interested in>	<SSI>
<KIT>	<is a>	<University>
<Student>	<is subtype of>	<Person>

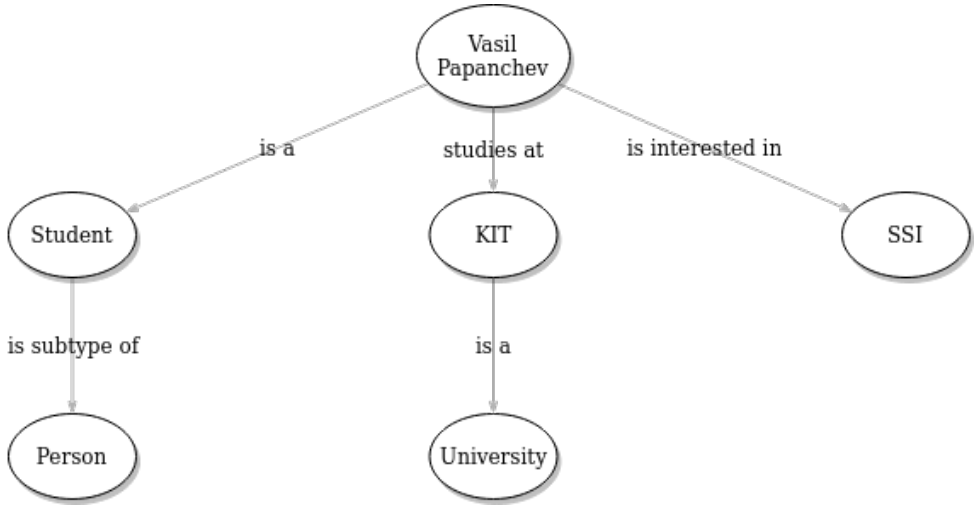


Figure 2.8: An example RDF graph

It is important to note, that RDF only defines a generic framework and a data model and not a specific syntax. There are multiple serialization formats for expressing RDF graphs:

- Turtle family of RDF Languages - N-Triples, Turtle, TriG and N-Quads
- JSON-LD - a lightweight Linked Data extension of the JSON format
- RDFa - syntax for embedding RDF into HTML documents
- RDF/XML - XML-based syntax for RDF

In the scope of this work, two of the serialization formats will be used: the JSON-LD syntax for representing Linked Data Credentials and the Turtle syntax for defining human-understandable and machine-readable Access Control Rules based on Verifiable Credentials (with Linked Data payloads).

2.5.2 SPARQL

The SPARQL Query Language [113] can be used to query and manipulate RDF graphs. SPARQL queries can range from simple pattern matching to complex queries containing value aggregation, path expressions and nested queries. There are three types of SPARQL queries:

- **SELECT Queries:** Traverse an RDF graph and return variables bindings conforming to the defined query. An example Select query which returns the names of persons and the number of their friends is given in listing 2.1.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name (COUNT(?friend) AS ?count)
WHERE {
  ?person foaf:name ?name .
  ?person foaf:knows ?friend .
} GROUP BY ?person ?name
```

Listing 2.1: An example SPARQL SELECT query

- **ASK Queries:** Traverse the graph and return a boolean result (true/false) showing whether the graph contains a solution to the query. An example is given in listing 2.2

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
ASK  { ?x foaf:name  "Alice" }
```

Listing 2.2: An example SPARQL ASK query

- **CONSTRUCT Queries:** Used to construct a new RDF graph specified by a graph template defined in the query. The graph is populated with triples conforming to the requirements defined in the query. An example CONSTRUCT query used to create an RDF graph containing vcard properties from an RDF graph with FOAF information is given in listing 2.3.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX ex: <http://example.org/>
CONSTRUCT   { ex:person#Alice vcard:FN ?name }
WHERE       { ?x foaf:name ?name }
```

Listing 2.3: An example SPARQL CONSTRUCT query

2.5.3 Shape Expressions (ShEx)

Shape Expressions (ShEx) [8] is a language for describing RDF graphs. The ShEx specification was published by the Shape Expressions Community Group in the form of a Draft Community Group Report which means that it is not an official W3C standard. However, we have considered using the ShEx language for modelling of Access Control Rules based on Linked Data Verifiable Credentials and have compared it to other alternatives. Therefore, a brief introduction to ShEx is given.

The ShEx language is used to define a ShEx Schema which represents a set of conditions that a given RDF graph should meet in order to be considered conformant, i.e., the ShEx validation of the given RDF graph using the ShEx Schema is successful. In this way, ShEx is intended for:

- Validation of RDF documents against the defined criteria
- Exchange of expected graph patterns
- Generation of User Interfaces
- Transformation of RDF graphs into other formats and structures

In the context of our SSI-based Access Control System, we have evaluated using the ShEx language for defining requirements on Linked Data credentials represented as RDF graphs. In this way, the access logic can be defined using Access Control Rules containing required credentials defined as ShEx Schemas.

A ShEx Schema contains a list of ShEx Data Shapes consisting of Node Constraints and Triple Constraints. Node constraints constrain the possible values for an RDF Term, while Triple Constraints define requirements on the RDF statements (triples) of the graph. Both are usually used in combination, e.g., a Triple Constraint defines that the RDF graph should contain some RDF statements with specific predicates, while the Node Constraints define requirements on the RDF Terms (more precisely: on the graph nodes representing these RDF Terms) that take the places of subjects and objects in these statements. An example ShEx Schema with a single Data Shape is given in listing 2.4. In the example, firstly, the Node Constraint *school:enrolleeAge* is defined which specifies RDF Terms of type *xsd:integers* in the range between 13 and 20. After that, a ShEx Data Shape, called *school:Enrollee*, is defined. It contains two Triple Constraints. The first one defines that RDF Terms conforming to the *school:Enrollee* Data Shape must play the subject role of an RDF statement with the *foaf:age* predicate. Furthermore, the object of this statement must be conforming to the *school:enrolleeAge* Node Constraint. The second Triple Constraint defines that the RDF Terms conforming to the *school:Enrollee* Data

Shape must play the role of the subject in exactly 1 or 2 RDF Statements with the predicate *ex:hasGuardian*. Furthermore, the object of these statements must be an RDF Term of type IRI.

```
PREFIX ex: <http://ex.example/#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX school: <http://school.example/#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

# Defines the Node Constraint school:enrolleeAge
# Terms conforming to this constraint are integers with 13 <= age <= 20
school:enrolleeAge xsd:integer MinInclusive 13 MaxInclusive 20

# ShEx Data Shape
school:Enrollee {
    # Triple Constraint - the objects of the foaf:age predicate must
    conform to the enrolleeAge Node Constaint
    foaf:age @school:enrolleeAge ;

    ex:hasGuardian IRI {1,2} # Another Triple Constaint
}
```

Listing 2.4: An Example ShEx Schema

Although, an RDF graph is validated against a ShEx Schema, it is still necessary to specify which nodes of the RDF graph should be validated against which ShEx Data Shapes defined in the ShEx Schema. This is done using a ShEx Shape Map, which can either directly specify which nodes are mapped to which Data Shapes (Fixed Shape Maps) or can use a more complex query which select graph nodes based on some of their properties (Query Shape Maps). For examples, see listing 2.5.

```
# Fixed Shape Map:
inst:Alice @ school:Enrollee,
inst:Bob @ school:Enrollee,
inst:Claire @ school:Enrollee,
inst:Don @ school:Enrollee

# Query Shape Map:
{FOCUS foaf:age _} @ school:Enrollee
# maps any RDF Term which is the subject of a foaf:age predicate
```

Listing 2.5: ShEx Schema

2.5.4 Shapes Constraint Language (SHACL)

The Shapes Constraint Language (SHACL) [66] is an official W3C Recommendation for validating RDF graphs against a defined set of conditions. The SHACL conditions are expressed as RDF graphs and are called "Shapes Graphs", while the RDF graphs which are being validated are referred to as "Data Graphs". The intended usages of SHACL include:

- Description and Validation of RDF graphs
- User Interface Generation
- Automated Code Generation

Because SHACL Shapes are defined as RDF graphs, SHACL Shapes can have global URIs, which means that any constraints defined in SHACL can be referenced by other SHACL graphs and shared on the Web as Linked Data. Furthermore, SHACL Shapes can be expressed using any RDF Syntax. In the scope of this work, we have used the Turtle serialization format to express SHACL Shapes.

As a brief introduction to SHACL, we will use a Turtle file 2.6 containing an example RDF Data Graph and a SHACL Shape (both taken from the official SHACL specification [66]) defining the following conditions:

- An instance of *ex:Person* can have at most one value for the property *ex:ssn*, and this value is a literal with the datatype *xsd:string* that also matches a regular expression.
- An instance of *ex:Person* can have any number of values for the property *ex:worksFor*. The values are IRIs and the RDF Terms identified by them conform to the SHACL Shape *ex:CompanyShape*.
- An instance of *ex:Person* cannot have values for any other property apart from *ex:ssn*, *ex:worksFor* and *rdf:type*.

Note that we have used the SHACL namespace `http://www.w3.org/ns/shacl#` which defines the RDF ontology used in SHACL. The RDF Term *ex:PersonShape* is of type *sh:NodeShape*, which is used when describing characteristics about specific graph nodes. By using the *sh:targetClass* predicate of the node shape, we define which nodes in the Data Graph should be validated against the *ex:PersonShape* shape. By using the *sh:property* predicate, we can reference further graph nodes by traversing the outgoing arcs of the targeted node, and then define more requirements on these nodes and on the properties of the RDF Terms represented by these nodes. For example, the first property in

ex:PersonShape constrains the values of any *ex:ssn* predicates by defining that there is a maximum one such predicate, its value is of type *xsd:string* and it matches the specified regular expression. The rest of the constraints are explained with comments in the listing.

```
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix sh:     <http://www.w3.org/ns/shacl#>
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#>
@prefix ex:     <http://example.com/ns#>

ex:Alice
    a               ex:Person ;
    ex:ssn          "987-65-432A" .
ex:Bob
    a               ex:Person ;
    ex:ssn          "123-45-6789" ;
    ex:ssn          "124-35-6789" .

ex:PersonShape
    a sh:NodeShape ;
    # Which RDF Terms to validate - all RDF Terms of type ex:Person
    sh:targetClass ex:Person ;
    # Max 1 ex:ssn predicate, its value is a string conforming to a regex
    sh:property [
        sh:path ex:ssn ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ;
        sh:pattern "^\\d{3}-\\d{2}-\\d{4}$" ;
    ] ;
    # The values of the worksFor property must conform to the ex:
   CompanyShape shape
    sh:property [
        sh:path ex:worksFor ;
        sh:class ex:CompanyShape ;
        sh:nodeKind sh:IRI ;
    ] ;
    # No other predicates besides ex:ssn, ex:worksFor and rdf:type
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .
```

Listing 2.6: Example SHACL Shapes Graph

Executing the SHACL validation on the provided RDF graph will return an unconforming validation result containing detailed information on all reasons for the unsuccessful validation, i.e., all RDF Terms that do not satisfy the defined SHACL Shapes. The exact format of the validation result is out of the scope for this work, but it is important to note, that the SHACL Validation Results are standardized and also expressed as RDF graphs - a clear advantage compared to the primitive validation results of the ShEx language.

Related Work

In this chapter, related work in the realm of Self-Sovereign Identity (SSI) and Web Access Control is presented. Section 3.1 lists the main technology leaders and working groups in the area of Self-Sovereign Identity (SSI) while in section 3.2 some scientific literature on SSI and Access Control is presented.

3.1 SSI Technology Leaders and Working Groups

The World Wide Web Consortium (W3C) [114] is an international community that develops open standards to ensure the long-term growth of the Web. The W3C Verifiable Credentials Working Group [120] is responsible for standardizing the way digital credentials are created, expressed, exchanged and verified securely on the Web. They are actively working on the, currently accepted as an official Recommendation, Verifiable Credential Data Model [105]. The W3C Decentralized Identifier Working Group [119] is responsible for standardizing the DID URI scheme and the data model and syntax of DID Documents.

The Rebooting of the Web of Trust (RWoT) [91] hosts meetings and workshops related to next-generation decentralized web-of-trust based identity systems. The goal of these meetings is to produce technical white papers that have a big impact on the future of digital identities.

The Internet Identity Work Group (IIW) [55] is another technical group that facilitates yearly meetings and workshops to discuss new ideas and solutions for digital identities. This work group was previously one of the Technology leaders and advocates for User-centric Identities [60].

The Decentralized Identity Foundation (DIF) [111] is a technical organization with the goal of creating and improving the foundations for an open and interoperable ecosystem for decentralized identity. They create technical specifications and reference implementations. Some of their core products include the DIDComm Messaging specification [30] - a protocol for secure communication on top of Decentralized Identifiers, the Peer-DID

Method Specification [32], and the Universal Resolver [34] - a driver-architecture system for resolving different types of DIDs.

The company Evernym [43] was founded in 2013 with the goal of "solving the digital identity crisis". It has been accepted as one of the Technology Leaders of SSI as it has co-founded the Trust-over-IP Foundation [116], the Decentralized Identity Foundation (DIF), the COVID-19 Credentials Initiative (CCI) [28], and the Good Health Pass Collaborative (GHPC) and are co-authors and co-editors of the Verifiable Credential and Decentralized Identifier Specifications hosted at the World Wide Web Consortium (W3C).

The Hyperledger Foundation [53] is a global collaboration, hosted by The Linux Foundation [112], focused on creating, maintaining and supporting open-source projects based on Distributed Ledger Technologies (DLTs) [84]. Hyperledger houses many libraries, tools and projects including such specifically build for SSI. Hyperledger Ursa is a cryptographic library used by many of the Hyperledger products for a secure and consistent cryptographic protocols. Hyperledger Indy is a public permissioned DLT purpose-built for Decentralized Identifiers. It was initially developed by Evernym and contributed to the Hyperledger Foundation. Indy implements the Indy DID Method and a Verifiable Registry for storing and managing schema definitions for Verifiable Credentials. Hyperledger Aries is a blockchain-agnostic library specifying and implementing many protocols for creating, transmitting and storing Verifiable Credentials. Aries defines the so-called *Aries Interop Profiles* which specify what protocols independent SSI Agents need to support in order to interoperate with each other.

The Sovrin Network [92] is an open-source SSI Framework originally developed by Evernym and later contributed to The Sovrin Foundation, a global non-profit organization. Sovrin is based on Hyperledger Indy and supports Decentralized Identifiers, Verifiable Credentials and Zero-Knowledge Proofs. Its Verifiable Registry is a public permissioned DLT governed by the Sovrin trust anchors.

COVID-19 Credentials Initiative (CCI) [28] is a global organization hosted by the Linux Foundation Public Health[1] (LFPH) trying to deploy open-source privacy-preserving projects using Verifiable Credentials for mitigating the spread of the COVID-19 virus.

[1] https://www.lfph.io/

3.2 Literature on SSI and Access Control

In this section, scientific literature related to SSI and Access Control is presented. First, section 3.2.1 introduces an Access Control framework called SSIBAC. SSIBAC is based on the XACML specification but uses Verifiable Credentials to model its access logic. Section 3.2.2 explains the Web Access Control (WAC) specification used in the Solid project and based on which the access logic of our system is designed. After that, section 3.2.3 presents Pattern-based Access Control - a mechanism to combine private project information with online data using semantic technologies in order to build an access control system. It is also based on the WAC specification and uses the SHACL language to model required credentials that the users need to provide. Although the used credentials are not based on the W3C standards, their modelling is useful and related to how we will model required credentials in our work. Finally, section 3.2.4 summarizes the architecture and possible deployment strategies of the, so-called Attribute Trust-enhancing Identity Broker (ATIB) - an Identity Provider architecture with the goal to simplify the interoperable adoption of SSI solutions for the service providers.

3.2.1 Self-Sovereign Identity Based Access Control

Self-Sovereign Identity Based Access Control (SSIBAC) [9] is an Access Control Model which uses Verifiable Credentials to model access control logic. SSIBAC is based on the XACML specification [46] but introduces the concept of *Permission Validators* which map Verifiable Credentials to attributes or roles, based on which, access control policies are defined. An SSIBAC instance consists of the following components:

- Set of users $U = \{u_1, u_2, ..\}$
  Each user is identified by a Decentralized Identifier (DID).
- Set of resources $R = \{r_1, r_2, ..\}$
- Set of issuers $I = \{i_1, i_2, ..\}$
- Set of verifiers $V = \{v_1, v_2, ..\}$ which represent the resource providers
- Set of Verifiable Credentials $L = \{l_1, l_2, ..\}$
- Set of Permission Validators $P = \{p_1, p_2, ..\}$
- Set of injective functions $\Psi = \{\psi_1, \psi_2, ..\}$ which map VCs to permission validators: $\psi_i : L \rightarrow P$

- An injective function $\chi : AC \rightarrow VP_R$ mapping access control policies to Verifiable Presentation Requests.

The procedure of approving or denying user access to requests in a system using an SSIBAC-based access control is the following: The user first requests a specific resource. Based on the access control policies (defined either as attribute-based or role-based XACML policies) and the defined χ function, the access control system responds to the user with a Verifiable Presentation Request. The user then provides their credentials packed in a Verifiable Presentation. The ACS verifies the credentials and validates whether the credentials map to the requested attributes/roles, thus approving or denying the user request.

The SSIBAC paper does not give any specific details on how the functions ψ and χ can be or have been implemented. However, one significant drawback of this model is that the set of possible user credentials is specified prior to the definition of those functions, i.e., this model requires credentials that are based on a specific schema published on the Verifiable Registry. Furthermore, specifying the required credentials prior to the initialization of the SSIBAC instance leads to a significant inflexibility of the access control logic.

To evaluate the performance of the SSIBAC model, the authors implemented the systems using Hyperledger Indy as a Verifiable Registry and Hyperledger Aries to create SSI agents capable of issuing, storing and providing Verifiable Credentials. It is not stated explicitly, but it is inferred that the type of credentials used are they Indy Anonymous credentials [64]. The evaluation was ran using the GreenLight Dev VON ledger[2]. Three Docker containers (issuer, user and access control system) were started on a virtual machine running Ubuntu 20.04 configured with 8 vCPUs and 32 GB memory. The authors split the entire process into three phases:

- Startup Phase: Register DIDs of the agents and publish credential schemas to the Verifiable Registry.
- Connect Phase: Connecting the SSI agents to each other and issuing the credentials to the user.
- Access Control Phase: The user requests a resource from the Access Control system. The ACS requests a presentation from the user. The user responds with a Verifiable Presentation. The ACS validates the received presentation and makes an authorization decision.

In the scope of this work, only the third phase will be regarded, as an access control system should be independent of the procedures of registering DIDs and issuing credentials, which,

[2] `http://greenlight.bcovrin.vonx.io/`

in a real-scenario, are executed completely separately from providing credentials to an access control system. The evaluated mean time for the access control phase of the implemented SSIBAC instance is said to be 0.9 seconds per request. This result is said to be independent of the number of credentials in the Verifiable Presentation. However, the evaluation was executed using only a single issuer. Therefore, the number of DIDs that need to be resolved for the verification of the credentials remains constant. In the evaluation of our system, different credentials issued by different issuers will be used, in order to provide more realistic evaluation results.

3.2.2 Web Access Control

Web Access Control (WAC) [23] is a W3C Editor's Draft of the Solid Community Group. It defines a decentralized access control system that can be used by Linked Data 2.5 systems to define the access authorizations to HTTP resources using the Access Control List (ACL) model. WAC is decentralized in the sense, that users, groups of users and resources are identified by URIs and users do not need to be registered on the Linked Data system in order to access the resources provided by it. That is, the owner of a resource, hosted on one site, can give access to users and groups of users hosted on different sites. WAC is the primary authorization mechanism used in the Solid Protocol [24].

To achieve a cross-domain understandable and re-usable access control logic, WAC defines a single RDF ontology (`www.w3.org/ns/auth/acl`) with which resource servers define their authorization rules. The fundamental unit of the WAC ontology is the *acl:Authorization* instance. It targets an *Access Object* (which represents a resource or a set of resources) and specifies the *access control modes* and *access subjects* for it. An Access Object can either be a single resource identified by its URI or a container for multiple resources in a hierarchical structure. There are four defined Access Control Modes - *acl:Read*, *acl:Write*, *acl:Append* and *acl:Control*. It is up to the resource owner (the server) to define how user requests map to the access modes. For example, an HTTP GET Request maps to acl:Read, while HTTP POST to an acl:Write, etc. Specifying the access subject of an authorization rule, i.e., who has access to the targeted resource, is done using the *acl:agent*, *acl:agentClass* and *acl:agentGroup* predicates, which specify a single subject (identified by its URI), a class of subjects or a group of subjects (identified by the `http://www.w3.org/2006/vcard/ns#Group` object). An example for a class of agents is *acl:authenticatedAgent* which gives access to authenticated users.

Example access control rules for two resources defined using the WAC specification is given in listing 3.1. According to these rules, any agent can read the *http://aifb.example.org/resources/r1* resource but only the user identified by the URI *http://aifb.example.org/people/vpapanchev* has write permissions to it.

```
@prefix acl: <http://www.w3.org/ns/auth/acl#> .
@prefix aifb: <http://aifb.example.org/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

aifb:authorizationRule1
  a                 acl:Authorization ;
  acl:accessTo      <http://aifb.example.org/resources/r1> ;
  acl:mode          acl:Read ;
  acl:agentClass    foaf:Agent .

aifb:authorizationRule2
  a                 acl:Authorization ;
  acl:accessTo      aifb:resources:r1 ;
  acl:mode          acl:Write ;
  acl:agent         <http://aifb.example.org/people/vpapanchev> .
```

Listing 3.1: Example WAC Authorization Rules

Achieving a Credential-based Access Control (combining WAC with Verifiable Credentials) was first suggested in the GitHub issues of the WAC specification[3]. From there, the discussion was moved to the Solid Authorization Panel[4]. One of the discussed solutions was to use Shape Expressions (see section 2.5.3) to define the required credentials. Although, no concrete realization of combining WAC and VCs was presented in these discussions, they have motivated our research on Access Control Rules based on VCs which will be presented later in this work.

3.2.3 Pattern-based Access Control

Pattern-based Access Control [121] aims to build an access control mechanism that combines private project information (originally in the realm of the building industry) with open online data by using semantic Web technologies. In this framework, a client is granted or denied access not based on their identity, but rather on confirmed by trusted third parties properties about them, such as being an employee of company X working in project Y or an inhabitant of a respective building. Expressing such properties in the default ACL implementation used in Solid, i.e., the WAC specification [23], would require assigning WebIDs (a type of URIs used to identify a person, organization, group or a device) to all clients and hardcoding them into the appropriate agent groups which is a

[3]Credential based access control (WAC + VC) - `https://github.com/solid/web-access-control-spec/issues/79`

[4]Solid Authorization Panel - `https://github.com/solid/authorization-panel/issues/79`

troublesome and inflexible approach.

Instead, Pattern-based Access Control extends the WAC specification to additionally describe RDF graphs which represent the required properties of the clients. For this, the SHACL language [66] is used. However, the properties are not modelled as Verifiable Credentials, but using nanopublications [48] - RDF documents containing an assertion, provenance and metadata. The nanopublications used in Pattern-based Access Control are digitally signed by some trusted issuers (identified by their WebIDs) and are called nanocredentials. They serve a similar purpose as Verifiable Credentials in the realm of SSI, but do not have a standardized data model are not inherently interoperable with DIDs. A specific motivation for using nanopublications in Pattern-based Access Control is not provided by the authors.

3.2.4 Attribute Trust-enhancing Identity Broker

Attribute Trust-enhancing Identity Broker (ATIB) [49] is an architecture to simplify integration by the service providers of SSI solutions used for authentication and authorization of the users. The architecture aims to provide a solution to the following challenges:

- **Multitude of SSI Solutions** - How should a service provider efficiently integrate support for SSI under the significantly fragmented landscape of technologically different SSI solutions?
- **Divergent Trust in Attribute Providers** - How can both the service provider and the users be enabled to integrate or use their preferred Identity or Attribute Providers (specific SSI ecosystems) while not forcing a single choice? Furthermore, how should supported providers be negotiated in a flexible way?
- **Existing Application Landscape** - How should a service provider integrate SSI solutions without introducing major changes to already existing applications?
- **Attributes based on Verifiable Claims** - In case the service provider wishes to assign attributes to its users, how should the service provider be enabled to easily issue verifiable claims to the users?

ATIB can technically be defined as an Identity Provider that provides traditional access management protocols as communication endpoints for Web applications. Internally, the access management is based on SSI by exchanging and verifying verifiable claims which represent and prove the identity attributes of the users. The authors evaluate three deployment strategies:

- **User Centric** - Each user deploys an ATIB instance themselves, thus, the ATIB instance runs within the trust boundary of the user and outside the trust domain of the service provider. In this trust model, the user defines the trusted Attribute Providers, i.e., the issuers of credentials. Therefore, it is unlikely that the trust requirements of the service providers will be satisfied, which makes this deployment strategy less practical.

- **Independent** - An independent entity deploys and hosts an ATIB instance. In this way, the deploying entity plays the role of a Trusted Third Party (TTP) which creates an additional trust boundary and also contradicts the one of the core ideas of SSI - eliminating Trusted Third Parties, as also specified by the Portability principle (see section 2.3.4).

- **Dedicated to the Service Provider** - The service provider hosts an ATIB instance and defines which Attribute Providers are trusted. In this way, the ATIB instance runs in the organizational trust domain of the service provider. User credentials are regarded as valid as long as they are issued by the approved by the service provider institutions. This deployment strategy reflects the SSI principles and is regarded (by the authors of [49] and by us) as most applicable.

In more details, the ATIB architecture consists of the following components:

- **Namespace Translator** - Because different SSI ecosystems use different names and identifiers for the same concepts, a way of translating between them is required for an interoperable internal operation. The Namespace Translator component translates attribute names, e.g., email, first name, last name, etc., specific to a single SSI solution into a consistent internal representation. In the presented example ATIB implementation, support for two SSI ecosystems is implemented - uPort [76] and Jolocom[5]. Both of them do not use a Linked Data approach to naming their attributes, which makes such translation of the attributes within verifiable claims required.

- **Trust Engine** - evaluates whether to trust a given verifiable claim of an identity depending on the Attribute Provider that issued this claim. The Trust Engine consists of one or multiple trust models which define trustworthiness of Attribute Providers and combinations of them.

- **Protocol Manager** - the core component of the ATIB architecture which implements various standard identity and access management protocols, e.g., OpenID Connect (OIDC) [98] and Security Assertion Markup Language (SAML) [90], for

[5]Jolocom Homepage - `https://jolocom.io/`

interaction with the Web application. The Protocol Manager uses the interfaces provided by the other components in order to retrieve, verify and evaluate the required user characteristics from the verifiable claims provided by the user. Following a successful evaluation of the verifiable claims, the characteristics are used in the subsequent protocols as usual user attributes.

- **Self-Sovereign Identity Manager** - provides multiple SSI interfaces which abstract from the supported underlying different SSI solutions. The provided interfaces can be used (by the Protocol Manager component) to create a new SSI identity, create an SSI challenge, verify an SSI challenge, request a verifiable claim, verify a verifiable claim and issue a new verifiable claim. For each supported SSI ecosystem, a wrapper around its corresponding APIs is required.

- **Verifiable Claim Issuer** - manages the issuance of verifiable claims to the users using their preferred SSI solution. Used in the cases that the service provider wishes to assign attributes to its users.

- **External Interfaces** - interfaces for administration of the ATIB interface, such as configuring supported SSI solutions or approved Attribute Providers, and communication with surrounding environment, e.g., retrieving information from other systems when issuing attributes to users.

The ATIB approach towards interoperability with multiple SSI ecosystems is based on implementing a separate wrapper for the APIs of each supported SSI solution. In our opinion, this is an infeasible approach due to the high number of existing SSI solutions. In this work, we will pursue interoperability based on usage of semantic technologies and the defined standards for Decentralized Identifiers and Verifiable Credentials and the usage of well-established SSI protocols for communication and exchange of credentials. However, the dedicated to the service providers deployment strategy and the combination of the SSI paradigm with the standard identity and access management protocols, such as OpenID Connect (OIDC) [98] and Security Assertion Markup Language (SAML) [90], are important contributions which can also be integrated into our Access Control System in a future work.

Analysis: Current State of Self-Sovereign Identity

This chapter analyzes different aspects of the current state of Self-Sovereign Identity (SSI). First, section 4.1 summarizes the most pressing challenges for the successful integration of SSI on the Web. Then, sections 4.2 and 4.3 show the high diversity within the underlying SSI technologies by analyzing multiple DID Methods and types of Verifiable Credentials and presents the major differences between them. In section 4.4, different approaches for achieving a secure communication on the Web are discussed, including the SSI approach of using the DIDComm Messaging protocol which uses the cryptographic materials stored in the users‘ DIDs. Finally, section 4.5 presents two important SSI technologies used during an exchange of credentials between a Verifier and a Holder.

4.1 SSI Challenges

In this section, important challenges for the successful integration of the SSI paradigm on the Web are discussed. The challenges presented here were summarized mainly from the work presented in [68] where the authors collected their data by conducting field research, including observing SSI ecosystems in a real context and conducting interviews with multiple domain experts in the area of SSI.

The Economics of SSI Adoption

Because SSI ecosystems require a fundamentally different infrastructure, transforming existing processes to run on SSI, or even only support SSI, is a risky strategy which requires a high upfront investment [68]. Therefore, a suitable business model that incentivizes service providers to adopt SSI should be designed parallel to the development of the SSI technologies. The authors of [68] give some example benefits for the organizations and institutions that integrate SSI in their products or services:

- More private and secure products and services

- Possibility for innovations
- Competitive advantage
- Improved brand and reputation
- Increased customer‘ retention, satisfaction, and customers‘ trust
- Development of strategic alliances and long-term relationships

Motivating End-Users to use SSI

As stated in [33], identity management is not the primary goal for users and the majority of them would not change their identity provider only due to better privacy protection. Users usually value their privacy, however, empirical and theoretical studies show that they are unwilling to invest time and resources in improving their security and privacy [1]. Therefore, a key challenge for the successful integration of SSI is to also provide a sufficient motivation or incentive for the End-Users.

Complexity of SSI and its User-Centric Design

In centralized user identity models, e.g., using an identity provider such as Facebook or Google, the users delegate almost all of the management of their identities to the provider, including the responsibility of protecting the identities and the privacy of the users, and enforcing the trustworthiness of the exchanged identity data. In this design, users exchange the control over their identities for a certain comfort - not having to manage the identities on their own. In contrast to this, SSI has a user-centric design, where users are in complete control over their identities which includes securing them. For more details about the different identity models, see section 2.3. The management of the Self-Sovereign Identities includes (on the technological level) complex procedures such as differentiating between public and private keys, properly securing private keys, managing possibly multiple DIDs and disclosing personal information in a privacy-protecting way using complex technologies such as Selective Disclosure and Zero-Knowledge Proofs. Failing to provide a user-friendly key management has been identified as one of the reasons for the unsuccessful realization of another identity-related technology - the PGP Web of Trust [123]. Thus, a key challenge for SSI is to abstract the details of the complex underlying technologies and provide user-friendly SSI applications, e.g., SSI wallets and automatic agents, so that SSI becomes sufficiently comfortable for the average user [31].

Horizontal Interoperability of different SSI Solutions

The two main building blocks of SSI are Decentralized Identifiers (DIDs) 2.4.1 and Verifiable Credentials (VCs) 2.4.2. Both, DIDs and VCs are defined as W3C Recommendations [104] [105] that only specify the data models and some requirements on the functionality of the technologies. The specifications deliberately allow for high degrees of flexibility in

the realizations of the technologies so that different implementations can be optimized for different use cases. Currently, there are 113 officially registered DID Methods [109] and 4 broadly defined flavours of Verifiable Credentials [127]. This high diversity within the SSI technologies is challenging for achieving functional interoperability between different SSI solutions for the same problems. The author of [127] also states that without a convergence toward a standard VC format and interoperable tools (such as programming libraries), functional interoperability will not be achieved. In [68], this challenge is defined as "Horizontal interoperability of different SSI solutions". Besides DIDs and VCs, other examples for SSI technologies where horizontal interoperability plays a significant role are the protocols used withing different SSI Ecosystems for securing communication and exchange of credentials.

Vertical Interoperability of different components of SSI Solutions

Besides Horizontal interoperability between different SSI solutions for the same problems, the authors of [68] also identify "Vertical Interoperability of different components of SSI solutions" as another important challenge. To overcome this challenge, SSI solutions should be as agnostic from the details of other underlying technologies as possible. For example, a protocol defining the exchanged messages between a Holder and a Verifier during an exchange of a Verifiable Credential should be independent of the formats of the exchanged credentials.

Lack of Understanding of SSI and Non-supporting Organization Culture

The challenge of building a common understanding of SSI withing an organization was identified as one of the most recurring challenges in the research of [68]. In a realistic scenario, integrating SSI in organizations requires funding and the commitment of the organization‘s top management. Because SSI is a new and complex topic, its benefits are often not clear to the executive leadership which leads to a non-supporting organizational culture.

4.2 DID Methods

As of the time of writing, there are 113 officially registered DID Methods in the W3C DID Specification Registries [109]. In our work, 4 types of DID Methods are defined depending on the type of the used Verifiable Registry (VR):

- **DID Methods without a VR:** Some DID Methods do not use any type of a VR and instead provide only locally-resolvable identifiers. Examples include the Peer DID Method [32] and the Key DID Method [75]. Such methods are beneficial in cases where fast and cheap DIDs can be used instead of the original DIDs of the

users. For example, if a user wants to remain anonymous, they can use a freshly generated Peer DID or Key DID which to be used for securing a communication to another SSI agent.

- **DID Methods using an existing DLT for a VR:** Most of the DID Methods use an already existing Distributed Ledger, such as Bitcoin or Ethereum, to anchor their Decentralized Identifiers. One of the benefits of this approach is that the DID Methods inherit the security properties of the used DLT and no additional governance network must be established. However, most existing DLTs do not inherently support Decentralized Identifiers which leads to an additional overhead in the encoding of the DID's data to be stored on the DLT. Furthermore, existing DLTs might introduce high costs for the necessary transactions for creating and updating DIDs.

- **DID Methods with their own VRs:** Some DID Methods, such as Sovrin[72], Indy[29] and KILT [5], implement their own distributed ledger purposely-built for Decentralized Identifiers. Such VRs are often well optimized for DIDs and allow for cheap *Create* and *Update* operations. However, a new governance network for the new DLT has to be established.

- **DID Methods with other types of VRs:** Some DID Methods do not use a DLT and instead use some already existing web technologies to anchor their identifiers. The did:web Method [47] can be used to anchor DIDs on existing web domains. To create such a DID, a web server with a published domain name has to be created. The web server provides the DID Document, while the DNS name of the server is encoded in the DID, e.g., *did:web:w3c-ccg.github.io:user:alice*. The did:dns Method [95] improves on this idea and anchors its DIDs directly on the Domain Name System infrastructure. The DID Documents of DNS DIDs are encoded as Resource Records that can be requested using the existing DNS infrastructure (after the DNS domain has been registered) without the need to create a web server.

The following DID Methods are presented in more detail as they will be used in the implementation.

4.2.1 Peer DID Method

As explain in section 2.4.1, in most cases, a DID is rooted in a public source of truth (also referred to as a Verifiable Registry), such as a Distributed Ledger, database or an alternative web technology. The Verifiable Registry serves as an immutable single source of truth and allows for the (authenticated) updates of the DID Documents and their global

resolvability [81]. Using such a DID Method for a Peer-to-Peer-oriented communication either requires using already existing DIDs of the peers or creating new DIDs. In the first case, long-lived identifiers are used for a short-lived peer-to-peer communication, thus sacrificing peer anonymity or pseudonymity [32]. In the second case, the creation of new DIDs introduces overhead and transaction costs for the publishing of the DIDs to this DID Method's Verifiable Registry. However, in most cases of a peer-to-peer communication, only the two communicating peers need to know each other's identifiers, public keys and service endpoints. Therefore, this global resolvability of the DID Method is not required and only introduces unnecessary costs and overhead [32].

The Peer DID Method [32] solves this issue by providing cheap, fast, scalable and secure DIDs that are independent of any Verifiable Registry by simply encoding the entire DID Document into the DID. Therefore, every entity with knowledge of the Peer DID can resolve it locally. There are different algorithms for generating a Peer DID, but the idea is simple - a so called *inception key* is used as a controller of the DID. The public counterparts of the inception key are included in the DID Document. Only the owner of the Peer DID knows the private parts of the inception key with which they can update the DID Document. The owner updates the DID Document by sending a delta message to the other peers. The delta message encodes the update of the DID Document, e.g., adding a new public key, and is signed using the inception key. This, of course, sacrifices the global-resolvability of the Peer DIDs. Nonetheless, Peer DIDs remain useful in many situations and have some notable advantages: [32]

- **No transaction costs**: all operations on Peer DIDs are independent of any Verifiable Registry and are thus completely free.
- **Scalability**: Peer DIDs are independent of any centralized entity or decentralized network and therefore scale entirely independent of any system's capacity.
- **Performance**: All operations on Peer DIDs (creating, updating, resolving) are fast, as they do not require connecting to a Verifiable Registry.
- **Privacy**: No personal information, such as public keys or service endpoints, need to be published on a Verifiable Registry that can be viewed by anyone. All information in the DID Documents is shared only to the parties that need to have access to it.

In the Peer DID specification [32], 5 layers of support for software systems working with Peer DIDs are defined:

- **Layer 1: Recognize**: Software systems offering support for Layer 1 need to understand what a Peer DID is and how it works. Requirements include checking validity of Peer DIDs and comparing Peer DIDs with each other.

- **Layer 2a: Accept Static Peer DIDs from others**: Software systems offering support for Layer 2a are capable of interacting with other entities that use Peer DIDs, as long as they do not attempt to update their DID Documents. The term *static* refers to exactly this - the Peer DIDs are generated once and not updated afterwards. This significantly simplifies the implementation overhead for this layer (and for Layer 2b), as no delta messages need to be processed and stored.

- **Layer 2b: Give Static Peer DIDs to others**: Software systems offering support for Layer 2b know how to generate Peer DIDs and send them to other systems. Similarly to Layer 2a, the Peer DIDs are not updated after the initial generation.

- **Layer 3a: Accept Dynamic Peer DIDs from others**: Software systems supporting Layer 3a are capable of interacting with other entities that use Peer DIDs and update them. This layer introduces significant overhead in comparison to layers 2a and 2b, as the software systems must implement procedures for receiving and understanding delta messages, verifying and storing them, running synchronization algorithms (are all delta messages received) and more complex resolution algorithms.

- **Layer 3b: Give Dynamic Peer DIDs to others**: Software systems offering support for Layer 3b are capable of using and updating their own Peer DIDs. This requires creating, storing and sending delta messages to other entities and running synchronization procedures with them.

In the scope of this work, Peer DIDs will be used to secure the communication between the client and the Access Control System. For simplicity, all implementations will only provide support for layers 2a and 2b as updating the Peer DIDs is not required for the goals of this work. In a real scenario, the Access Control System would benefit from implementing support for Layer 3a, as in doing so, it will be interoperable with more complex clients that also want to update their Peer DIDs.

4.2.2 ETHR DID Method

The Ethr DID Method [118] utilizes the ERC1056[1] pattern which suggest the deployment of a Smart Contract for a lightweight management system for creating and managing identifiers for an off-chain usage in a DID-compliant way. ERC1056 has been approved and deployed to the Ethereum main network and many of its test networks, including Rinkeby and Ropsten. It is also deployed to the Polygon network - a layer 2 solution built on top of Ethereum[2].

[1] `https://github.com/ethereum/EIPs/issues/1056` - ERC: Lightweight Identity

[2] `https://github.com/uport-project/ethr-did-registry#contract-deployments` - Ethereum DID Registry

Creating an Ethr DID is based on creating a fresh Ethereum address which only requires the offline generation of a secp256k1 key pair, thus, getting a new Ethr DID is completely free. Resolving an Ethr DID uses the read-only functions and contract events on the ERC1056 registry. The registry stores contract events that can be used to update Ethr DIDs. There are 3 types of contract events:

- **DIDOwnerChanged:** Changing the controller of the DID
- **DIDDelegateChanged:** Adding or revoking delegate keys which represent additional Ethereum addresses that can sign transactions or even authenticate instead of the controller of the DID.
- **DIDAttributeChanged:** Adding Non-Ethereum attributes, such as public keys and service endpoints, to the DID Document. For example, to add a new Ed25519 Verification Key to the DID Document of the DID *did:ethr:mainnet:0xf3beac30c498d9e26865f34fcaa57dbb935b0d74*, a *DIDAttributeChanged* event for the account *0xf3beac30c498d9e26865f34fcaa57dbb935b0d74* with the name *did/pub/Ed25519/veriKey/base58* and the value of *0xb97c30de767f084ce3080168ee293053ba33b235d7116a3263d29f1450936b71* (the base58 encoding of the Ed25519 public key) needs to be created (by sending a signed transaction to the ERC1056's deployed smart contract).

Using the Ethr DID Method has some notable advantages [118]:

- No transaction fees when creating new Ethr DIDs
- Creating new identifiers is done entirely offline and is completely private.
- Ethr DIDs can be rooted on the Ethereum main network which is one of the most reliable and trusted decentralized networks.
- The ERC1056 Smart Contract has been deployed to many of the Ethereum test networks. Developers of SSI applications can work with Ethr DIDs for free.
- Ethr DIDs can be deployed to any EVM-based network.

4.2.3 Indy DID Method

The Indy DID Method [29] utilizes the Hyperledger Indy Ledger which is a public permissioned distributed ledger specifically built as a Verifiable Registry for Decentralized Identifiers and Verifiable Credentials. There are multiple instances of a deployed Hyperledger Indy network, e.g., the 3 Sovrin[3] networks (BuilderNet, StagingNet, MainNet), the

[3] `https://sovrin.org/overview/` - Sovrin Networks

IDUnion Indy Network[4] or the GreenLight VON Network[5]. Indy DIDs have the following syntax: *<did:indy:namespace:namespace identifier>*. The *Namespace* identifies the specific Indy network, while the *Namespace Identifier* is the unique identifier of the DID which in most cases is derived from the initial Verification Key associated with this DID. Some examples of Indy DIDs are:

- **did:indy:sovrin:7Tqg6BwSSWapxgUDm9KKgg** - A DID with unique identifier *7Tqg6BwSSWapxgUDm9KKgg* on the main Sovrin network.
- **did:indy:sovrin:staging:6cgbu8ZPoWTnR5Rv5JcSMB** - A DID with unique identifier *6cgbu8ZPoWTnR5Rv5JcSMB* on the Staging Sovrin network.
- **did:indy:idunion:test:2MZYuPv2Km7Q1eD4GCsSb6** - A DID with unique identifier *2MZYuPv2Km7Q1eD4GCsSb6* on the testing IDUnion network.

Creating and updating Indy DIDs is done by writing a **NYM** transaction to the ledger. NYM stands for 'Verinym' and is associated with the legal identity of the owner of an Indy DID. Depending on the specific ledger, NYM transactions might only be written to the ledger by specific nodes with the proper permissions and roles, such as a Transaction Endorser. An Indy NYM transaction consists of a destination identifier, an ED25519 verification key, an optional JSON object representing some content of the DID Document, such as service endpoints and additional keys, and additional metadata, such as the version of the NYM transaction. NYM transactions need to be signed by the controller of the DID identified by the NYM transaction's destination. In simple terms, a NYM transaction is the data structure stored on the Indy ledger that represents a DID Document. To resolve a DID, the most recently written NYM transaction needs to be translated to the format of a DID Document as defined in the DID Specification and returned to the client.

In 2018, the Province of British Columbia deployed the first Hyperledger Indy network under the name of Verifiable Organizations Network (VON). The source code of the network including a Ledger Browser have been contributed[6] and can be used to deploy a test Indy network locally which is very well-suited for early development stages.

4.2.4 KILT DID Method

KILT [65] is an SSI ecosystem consisting of the KILT Blockchain, client applications and decentralized services built using the KILT Software Development Kit[7] (SDK) which implements the KILT Protocol. The KILT Protocol defines the 3 main layers:

[4] `https://explorer.idu.network/` - IDUnion Indy Network
[5] `http://greenlight.bcovrin.vonx.io/` - The GreenLight VON Network
[6] `https://github.com/bcgov/von-network` - VON Network Github
[7] `https://github.com/KILTprotocol/sdk-js` - KILT Javascript SDK

- Data Models: The data formats of KILT identities, claim definitions, attestations, verifications and others.
- Messaging Protocols: Rules, guidelines and workflows for the communication between the different roles (Attester, Claimer, Verifier) in the ecosystem.
- Blockchain Nodes: Functionality of the nodes supporting the underlying distributed ledger of KILT.

KILT uses the KILT Blockchain as an underlying Verifiable Registry. The KILT Blockchain is a public permissionless blockchain built using the Parity Substrate blockchain framework[8] and uses a Limited Delegated Proof-of-Stake (LDPoS) consensus algorithm, which is a type of Proof-of-Stake consensus used by KILT. With LDPoS, any node can use its KILT Coins to back a specific collator that they trust in the system. The KILT Blockchain has been deployed as a parachain to the Polkadot network.

There are two types of KILT DIDs - Light DIDs and Full DIDs. A Light KILT DID has the structure <***did:kilt:light:key-encoding _ did-identifier(:additional-detals)***>, where the *did-identifier* is a base58 encoded KILT address. Creating Light KILT DIDs is done entirely off-chain and is free, however, they cannot be updated and do not provide high security properties, as they are not rooted to any single source of truth. Full KILT DIDs have the syntax <***did:kilt:did-identifier***> and are rooted on the KILT Blockchain. Full DIDs can be updated and can store multiple authentication and encryption keys and service endpoints. Some operations, such as issuing a credential, can only be executed using Full DIDs.

The KILT Protocol defines 3 roles - Attester, Claimer and a Verifier, which are semantically equivalent to the 3 roles defined in the W3C Verifiable Credentials Data Model [105] - Issuer, Subject/Holder and Verifier. The credentials of KILT are called *Claims*. A Claimer requests a Claim from an Attester, which verifies the request and attests the claims of the Claimer by issuing them a verifiable Claim. Later, the Claimer can use this Claim to prove the attested information about them to other Verifiers. A KILT Claim is always attested based on a specific *CType* (short for "Claim Type"). A CType plays the role of a claim schema which defines the data model of the claims of this type. CTypes are published on the KILT Blockchain and can be reused by multiple Attesters. The White Paper of the KILT Protocol goes into much details about the importance of standardization of claim types for interoperable applications. To incentivize different Attesters to re-use standardized CTypes, the publishing of new CTypes includes paying transaction fees. However, this interoperability and standardization of the KILT credentials is only presented in the local context of the KILT Ecosystem.

[8] `https://www.parity.io/technologies/substrate/` - Parity Substrate Blockchain Framework

To evaluate the interoperability of KILT with other SSI ecosystems, we have tried to use KILT DIDs and Claims in our Access Control System, which turned out to be an infeasible in overhead task. KILT CTypes are neither based on the W3C Verifiable Credentials Data Model, nor are they Linked Data. Furthermore, when using the KILT SDK, which provides the full functionality of the KILT Protocol, such as connecting to the KILT Blockchain, creating KILT DIDs, publishing CTypes, issuing attestations, proving and verifying Claims, and others, it is impossible to access the Private Keys controlling the KILT DIDs. Instead, they are only stored in the, provided by the SDK, Polkadot-based Wallet implementation and cannot be used outside of the KILT context[9]. In our view, such a design decision clearly contradicts the fundamental SSI principles - Existence, Control and Access 2.3.4. Therefore, we evaluate the interoperability of the KILT Protocol to be very poor and have opted out of using KILT DIDs in our implementation.

4.2.5 Web DID Method

DID Methods that use a Distributed Ledger as a Verifiable Registry face the challenge of bootstrapping sufficient meaningful and trusted data around identities in order to be massively adopted. Web DIDs [47] solve this problem by introducing the simple concept of publishing DIDs on well-established and trusted domains. The DID Document of a Web DID is hosted on a publicly accessible domain and can be retrieved over standard web technologies, such as HTTP and DNS. Such DIDs inherit the security and trust properties of the domains on which they are published.

To register a Web DID, a DID Document is created as a *did.json* file and stored either under the well-known URL of the domain or, if the domain will provide multiple DIDs for different users, under a specific path on this domain. An example Web DID as the single DID of a domain would look like `did:web:aifb.kit.edu` and its DID Document would be accessible under `https://aifb.kit.edu/.well-known/did.json`. Alternatively, the Web DID might also contain a specific path to where the DID Document is stored, e.g., the DID Document of `did:web:aifb.kit.edu:employees:papanchev` will be accessible under `https://aifb.kit.edu/employees/papanchev/did.json`.

4.2.6 DIF Universal Resolver

As explained in the previous sections, DID Methods can differ significantly in the way a DID is resolved to a DID Document. However, once a DID has been resolved, due to the standardized data model of the DID Documents, it is significantly easier to build an

[9] `https://github.com/KILTprotocol/sdk-js/issues/501` - KILT SDK GitHub Issue: Accessing Private Keys of DIDs

SSI application that is interoperable with different DIDs. For example, assuming that the DID has already been resolved, the procedure of accessing a specific public key of a DID, should be agnostic of the DID Method.

In 2017, the Decentralized Identity Foundation (DIF) [111] initiated work on a new project, called *Universal Resolver* [94]. The goal of the Universal Resolver is to provide an extensible framework for resolving DIDs of different DID Methods. The Universal Resolver provides a unified interface that takes a DID as input and returns a resolution object which contains the DID Document (if the resolution was successful) and some additional metadata. Internally, the Universal Resolver is a driver-based architecture, with each driver being responsible for resolving the DIDs of a specific DID Method. Drivers are expected to be implemented and contributed by the communities implementing the different DID Methods. Currently, there are drivers for over 30 DID Methods, including did:sov, did:btcr, did:key, did:web, did:ethr, did:kilt and did:indy. The Universal Resolver and the provided drivers are open-source [34]. DIF provides two deployed instances for test purposes, but users can also deploy a local instance of the resolver and configure supported DID Methods.

4.3 Flavors of Verifiable Credentials

The VCs Data Model W3C Recommendation [105] defines a universal data format for expressing and presenting claims in a cryptographically secure, tamper-evident, privacy-respecting, and machine-verifiable way. Such a data format allows for an interoperable credential definition and processing. However, multiple different "flavors"[127] of Verifiable Credentials have emerged, each using a different syntax, cryptographic proofs and verification methods. This has complicated the functional interoperability across SSI ecosystems due to diverse SSI protocols and incompatible programming libraries. In this section we provide an overview of the four primary flavors of VCs and discuss the differences in their processing and verification:

1. **JSON-LD + LD Signatures:** VCs expressed in Linked Data syntax using Linked Data Signatures [74] as proofs
2. **JSON-LD + BBS+ Signatures**: VCs expressed in Linked Data syntax using BBS+ Signatures [73] which also allow for Zero-Knowledge Proofs
3. **JSON-JWT**: VCs expressed as JSON Web Tokens [58] using JSON Web Signatures [57] as proofs. The VC's payloads can be both pure JSON or JSON-LD.
4. **JSON-ZKP-CL**: Zero-Knowledge Proofs Anonymous Credentials (AnonCreds) [64]. This credential format, originally developed by the companies Evernym [43] and

Sovrin [92] and then contributed to the Hyperledger Indy [53] codebase, tries to additionally protect the Holder‘s privacy by not including any interlinkable Holder identifiers in the credentials. Anonymous credentials have originally been implemented using the Camenisch-Lysyanskaya Digital Signatures scheme [22] but have recently also been combined with BBS+ Signatures.

According to its designers, the JSON-LD + BBS+ Signatures should not be seen as a different format of Verfiable Credentials. This does make sense as the BBS+ Signatures are said to be compliant with the Linked-Data Proofs specification [74]. However, it is also important to note, that using JSON-LD + BBS+ Signatures credentials requires specific public/private keys generated on the BLS12-381 pairing-friendly eliptic curve and specific algorithms for their verification.

Semantic Meaning of Credential Attributes

The first encountered difference between the VC flavors occurs when the Verifier tries to understand the semantics of the content inside the credentials.

- **JSON-LD Credentials** use a Linked-Data context that references RDF-defined schemata published somewhere on the Web. In this way, a Verifier can easily check the meaning of any attribute of a credential by only accessing the referenced ontologies.
- Some attributes of **JSON-JWT Credentials**, such as issuer and expiration dates, are standardized in the IANA-registered schema for JWT[10]. Otherwise, the JWT Credentials can either encode pure JSON or JSON-LD payloads. In the case of pure JSON payloads, the claim-specific attributes are not standardized in any way and the Verifier must use some Out-of-Band mechanism to infer their semantic meaning.
- Every **JSON-ZKP-CL Credential** references a specific credential definition schema stored on the same distributed ledger used for anchoring the Decentralized Identifiers. In this way, the Verifier can infer the meaning of the credential's attributes. However, an additional overhead for publishing and reading credential definitions to/from the DLT is required.

Anchoring Credentials to their Holders

Some VC flavors also differ in how the subjects of credentials are specified, i.e., identifying the entity about which claims are issued.

- **JSON-LD with LD-Proofs and JWT Credentials** include some identifier of the subject, e.g., a DID. The Holder has to prove ownership of the included DIDs when presenting the credentials.

[10]JWT standardized claim names - `https://www.iana.org/assignments/jwt/jwt.xhtml`

- **JSON-LD with BBS+ Signatures Credentials** allow for either including a DID or a Public Key as an identifier of the Holder. The Holder has to prove ownership of either the DID or the corresponding private key when presenting the credential.
- **JSON-ZKP-CL Credentials** are anchored to a so-called "Link Secret" only known to the Holder. A cryptographic commitment of the Link Secret is added into the credential which is later used by the Holder to prove ownership of the credential.

Creating and Securing the Credentials

Before sending the credential to the Holder, the Issuer has to pack the claims in a specific order and has to create the needed proofs with which the Holder can later prove the integrity of the credential to the Verifiers.

- To create **JSON-LD Credentials** the Issuer first dereferences the attributes (fetching their complete names) and canonicalizes them (putting them in an alphanumeric order). When using LD-Signatures, the entire credential is signed, while with BBS+ Signatures each attribute is signed separately.
- When using **JSON+JWT** the entire credential is signed without using a specific ordering of the attributes.
- With **JSON-ZKP-CL** Credentials, a numeric representation for each attribute is generated and then both the text and the numeric representation of each attribute is signed separately with a CL-Signature.

Creating a Verifiable Presentation

The next divergence between the VC flavors occurs when the Holder wants to share their credentials to a Verifier and needs to create a VP. In all cases, the Verifier provides a nonce to be used by the Holder to prevent Replay-Attacks.

- With **JSON-LD + LD-Proofs** and **JSON-JWT** credentials, the Holder packages the entire unmodified credential together with some metadata and the Verifier's nonce in a VP, digitally signs it and sends it to the Verifier.
- With **JSON-LD + BBS+Signatures** credentials, the Holder can choose which of the attributes of the credential to disclose, by only packing them in a VP. This is possible, as each attribute is signed separately by the Issuer when creating the credential.
- The Holder of a **JSON-ZKP-CL** credential creates a new credential that only reveals some of the attributes of the original one, optionally with some additional

predicates (assertions such as greater than a specific value). The Holder then creates the proof for this new credential using Zero-Knowledge algorithms that prove knowledge of a VC that has been issued by the specific Issuer and contains the included attributes. This proof is also bound to the same Link Secret to which the original credential is anchored, thus proving ownership of the credential.

Verification of the Presentations

The last step is for the Verifier to verify the integrity and authenticity of the received credentials. In all cases, the Verifier resolves the Issuer's DID and accesses their public key.

- In the case of **JSON-LD + LD-Proofs** and **JSON-JWT** credentials, the Verifier resolves both the Issuer's DID and the Holder's DID (both are included in the VC) and accesses their public keys. The Verifier uses these public keys to verify the signatures of the VC and VP.
- With **JSON-LD + BBS+ Signatures** credentials, the Verifier needs to check the signature for each contained attribute separately again using the public keys of the Issuer and Holder.
- With **JSON-ZKP-CL Credentials**, the Verifier verifies the Zero-Knowledge Proofs for each contained attribute separately using the public key of the Issuer and the credential definition schema fetched from the Verifiable Registry.

The differences between the four defined flavors of VCs are summarized in table 4.1.

Credential Format	JSON-LD + LD-Signatures	JSON-LD + BBS-Singatures	JSON-JWT	JSON-ZKP-CL (AnonCreds)
Attributes Semantics	RDF contexts	RDF contexts	RDF Context / Out-of-Band	On-ledger Credential Schema
Holder Binding	Holder‘s DID included in VC	Holder‘s DID included in VC	Holder‘s DID or Public Key included in VC	Link Secret and cryptographic commitments
Presenting Credentials	Pack entire VCs in a VP + digital signature	Pack some attributes of VCs in a VP + digital signature	Pack entire VCs in a VP + digital signature	Prove knowledge of an issued credential containing some of the attributes
Verification	Resolve DIDs of Holders and Issuers, verify signatures of VCs and VP	Resolve DIDs of Holders and Issuers (if DIDs used), verify signatures for each attribute	Resolve DIDs of Holders and Issuers, verify signatures of VCs and VP	Resolve credential schema, verify the ZKP-based proof for each attribute

Table 4.1: Summary of Credential Flavors

4.4 Secure Communication on the Web

This section gives an overview of different mechanisms for achieving a secure communication on the Web. Firstly, the well-established but centralized approach of Public Key Infrastructures [87] is presented. Then the approach of *PGP Web of Trust* [128] is introduced and the reasons for its unsuccessful realization briefly discussed. After that, the decentralized counterpart of PKIs - Decentralized Public Key Infrastructure (DPKI) [27], is explained. The section concludes by showing the suitability of Decentralized Identifiers and Verifiable Credentials for the establishment of a secure communication.

4.4.1 Public Key Infrastructures

With asymmetric cryptography [36] Alice and Bob can communicate securely and privately as long as Alice knows her private key and Bob's public key and vice-versa. However, how can Alice be sure that a specific public key indeed belongs to Bob and not to the malicious Malory impersonating Bob? The majority of secure communication on

the Web nowadays relies on some Public Key Infrastructure (PKI) [87] facilitating the bindings between public keys and identities. With PKIs one communicating party can be sure of the identities of other systems with which it communicates. For example, if Alice wants to send a message to Bob encrypted with Bob's public key, the PKI ensures Alice that this specific public key indeed belongs to the identity Bob. However, this certainty only holds assuming two core security properties of the PKI: *accurate registration* - a user can register only identities belonging to themselves, and *identity retention* - a user can never impersonate an identity already registered by another user [45]. We will show that both of these properties have been violated in the past.

The most popularized form of a Public Key Infrastructure is the hierarchy of Certificate Authorities (CAs) - trusted entities that issue (digitally sign) certificates (for example X.509 certificates [82]) to users who can then prove ownership over their own public keys. In this way, Bob can provide a certificate showing which public key is the real "Bob's public key" and a digital signature proving control over this public key. As long as Alice trusts the CA that issued Bob's certificate, Alice will be convinced of Bob's identity when communicating with him. To scale this infrastructure to the billions of identities on the Web, a hierarchy of CAs with transitive trust was introduced - a root CA can authorize another CA to issue certificates or even authorize other subordinate layers of CAs.

In simpler terms, a PKI can be seen as a complex trust graph of CAs responsible for the management of a huge database mapping public keys to identities. Because every CA is capable of adding new entries to this database, a PKI is only as secure as its weakest CA [40]. This leads to centralized Single Points of Failures that have been exploited numerous times. The most notable example of a malicious behavior due to failure of a PKI is the infiltration of the Dutch CA "Diginotar" in 2011 [89] leading to the malicious issuance of certificates for multiple Google domains. This allowed for a large scale Man-in-the-Middle attack on over 300,000 users located in the Islamic Republic of Iran [117]. Another notable example is the TrustWare incident [41] where subordinate root certificates have been sold to private companies with the official goal to spy on their employees internet traffic. One could argue that this is already a security flaw on its own. However, not only was the privacy of the employees of these companies endangered, but also the issued root certificates could have been used by the private companies to act as official CAs themselves and potentially issue invalid certificates for any domains [45].

4.4.2 PGP Web of Trust

Pretty Good Privacy (PGP) [128] is a software program with the goal of making strong cryptographic algorithms available to use to the average uneducated in computer science person. It aimed to achieve this by providing easy-to-use algorithms for encryption and

decryption of messages and files, and creation and verification of digital signatures for authenticated communication. PGP is based on Public-Key cryptography but tries to solve the problem of binding public keys to identities in a decentralized fashion. Instead of having explicit centralized authorities (CAs) trusted by all users, PGP lets its users define on their own whom to trust. People who know each other can digitally sign each other's public keys. In this way, one user certifies their belief in the binding of the signed public key to the certified owner. The resulting graph of public keys connected by digital signatures is referred to as the "PGP Web of Trust" [25]. In this type of PKI, Alice will trust Bob's public key if it contains a digital signature created by someone else whose key is already trusted by Alice.

However, some flaws of the PGP approach and its implementations have led to its incomplete establishment on the Web. Nowadays, even the main PGP use case - encrypting email communications, has almost entirely been replaced by modern End-to-End encryption protocols as security professionals are switching to messaging applications such as Signal and Telegram[11]. One of the main reasons for the failures of PGP is its insufficiently user-friendly interface leading to security problems such as exposing private keys in plaintext emails or general failure of securing communication [122],[123]. Secondly, the strict and limited in expressiveness format of PGP certificate attributes [21] leads to ambiguous meaning of attribute values and thus poor interoperability. A more semantic approach and extensible formats adhering to the Linked Data principles [15] could have been the better option.

4.4.3 Decentralized Public Key Infrastructure

In 2014, Certcoin - a completely Decentralized Public Key Infrastructure (DPKI) was presented [27]. Certcoin uses a blockchain as a distributed ledger for managing a public database binding public keys to identities (domain names). Using a consensus algorithm the ledger is securely (assuming honest majority) managed by a network of nodes which are financially incentivized to honestly validate new registrations and updates of public keys before accepting them on the ledger. Although still not globally adopted, Certcoin has proven the suitability of blockchain technologies for achieving an infrastructure for a secure communication on the Web. The Certcoin protocol has also been adapted to a privacy-aware blockchain-based PKI to avoid the public binding of private identities to public keys and instead allow users to decide themselves when and to whom to disclose their identities [6].

[11] https://www.cryptologie.net/article/487/a-history-of-end-to-end-encryption-and-the-death-of-pgp/

4.4.4 Communication over Decentralized Identifiers

In simple terms, DPKIs utilize a decentralized public ledger to manage a database mapping identities to public keys and provide efficient lookup operations. Decentralized Identifiers (DIDs) take this one step further, as the information stored on the verifiable registry is resolvable to an extensible document containing more information than just public keys, e.g., service endpoints and supported communication protocols, thus achieving one of its key use cases - *Service Endpoint Discovery*[12]. Furthermore, the DID Subject can prove their identity and many other properties about themselves (using the Verifiable Credentials issued to their DIDs) to their communication partners. This suitability of the SSI infrastructures to act as a decentralized PKI has led to the creation of the DIDComm Messaging specification.

DIDComm Messaging [30] is a formal specification developed and managed by the Decentralized Identity Foundation [111]. DIDComm Messaging is a message-based, connectionless and asynchronous protocol used for encrypting and authenticating messages using the cryptographic materials and service endpoints contained in the DID Documents of the communicating parties. In this way, DIDComm Messaging inherits the trust properties of the underlying DID Methods and does not require centralization for the mapping of cryptographic keys to identities. DIDComm Messaging also specifies a simple protocol for packing, unpacking and forwarding messages over intermediate systems responsible for the delivery of the messages to their recipients. In the simpler case of direct communication without mediators, DIDComm Messaging can be seen as a specification of encrypting and authenticating messages based on Decentralized Identifiers. In order for a DID to be used for DIDComm Messaging, its resolved DID Document must be updated to contain the *DIDComm Messaging Service Endpoint* which specifies the service endpoint for the transport of DIDComm messages, the accepted message types and optionally routing keys used to pack messages when routing through mediator systems.

As mentioned, in order to use DIDs for communication, their DID Documents must be updated to contain the appropriate service endpoints, which does have some drawbacks. Firstly, the update-operations of the DID Documents in many DID Methods, e.g., did:btcr [4] and did:ethr [118], require write transactions on the underlying verifiable registry which introduces additional transaction costs. Furthermore, due to the public nature of the DID Documents, the service endpoints will be accessible by everyone. In the case of a direct, pairwise communication between two parties, Alice and Bob, it makes sense that only Alice and Bob need to know how to reach each other by directly exchanging their service endpoints, instead of posting them to a persistent public storage.

Peer DID [32] is another DID Method conforming to the official DID Specification [30]

[12]https://www.w3.org/TR/did-use-cases/#serviceEndpoint

that solves the issues mentioned above. As explained in section 4.2.1, Peer DIDs provide fast, cheap and scalable DIDs that are independent of any central source of truth, i.e., without utilizing a verifiable registry. Instead, Peer DIDs are resolvable DIDs by encoding the entire DID Documents in the identifiers. Thus, creating new Peer DIDs is free as no transactions on the registry have to be written, instead Peer DIDs are stored and managed locally by their owners. A user can create a fresh Peer DID for each separate communication the user undergoes. After a secure DIDComm Messaging communication channel on top of Peer DIDs has been established, the communicating parties can prove to each other control over their "real", long-term DIDs, in order to establish authenticated communication. In this way, the original DID Documents do not need to be updated and the service endpoints of the communicating parties are not publicly published, therefore, solving both of the previously defined issues.

4.4.5 DIDComm Messaging and Man-in-the-Middle Attacks

The DIDComm Messaging protocol [30] aims to achieve a secure, i.e., tamper-proof and confidential, communication pipeline between two owners of DIDs. This should not be confused with a secure communication between two authenticated identities. Using the DIDComm Messaging protocol guarantees that only the controller of the DID to which an encrypted message was sent, can decrypt it and access the plaintext of the message. Whether that DID indeed belongs to the intended identity is, currently, not in the scope of the protocol and should be validated by the upper layers in order to prevent Man-in-the-Middle attacks (see section 5.1.6 in [30]).

A Man-in-the-Middle (MITM) attack [80] is a popular attack type in cryptography in which the communication between two parties (Alice and Bob) is secretly relayed through a malicious adversary (Mallory). Although, Alice and Bob believe to communicate directly with each other, Mallory has positioned herself in between them and intercepts all of the exchanged messages, either to eavesdrop on the communication or to even modify outgoing messages undetected [78]. There are different ways of executing a MITM attack mostly depending on the targeted communication channel. The most popular and massive example for a MITM attack is the malicious spying on over 300,000 Iranian users in which the communication between the users and some Google services was relayed (allegedly by an infiltrated DNS server) through an eavesdropping server impersonating Google using maliciously issued certificates by the hacked Dutch Certificate Authority Diginotar [89]. A simpler version of the attack is to have a direct access to a wireless router and spy on a not End-to-End secured communication of users connected to the router [78]. In the context of an Access Control System, a MITM attack can be used to impersonate a user and gain unauthorized access to resources or to impersonate the ACS and steal user

credentials.

MITM attacks related to the DIDComm Messaging protocol have been discussed and criticized on multiple occasions. In the official specification of the protocol [30], only a single sentence on the topic is given which states that the provided by the DIDComm Messaging guarantees have often been misunderstood. The reader is then forwarded for more information to an (as of time of writing) empty section of the DIDComm Guidebook[13]. A more comprehensive discussion on the topic of how and whether at all MITM prevention should be addressed in the specification of the protocol can be found in a GitHub issue[14] in the specification repository. The final result of the discussion is that the attack should be prevented by the upper layers, e.g., using Verifiable Credentials (VCs). The danger of MITM attacks and some methods for their prevention when using the DIDComm Messaging protocol together with Peer-DIDs are also discussed in the Peer DID Method specification ([32], appendix A - Security Considerations). There, two approaches for MITM prevention are presented - using a Trusted Third Party to validate the identities of the communication parties or using VCs.

With VCs, users can prove to their communicating partners that the DIDs they are currently using for securing the communication indeed belong to the intended identities. There are two realizations of this approach depending on whether the *Prover* (the party that is currently proving its identity) is using their original DID, i.e., the DID to which the VCs of the Prover are linked, or another DID, e.g., a freshly generated Peer-DID, for the communication. In the first case, simply presenting a VC that certifies the identity in control of the DID is sufficient. In the second case, the Prover needs to provide both, a VC stating their identity, and a proof linking the communication DID with the DID to which that credential is issued. In more details, the Prover sends a VP including two credentials:

- Prover‘s Identity - a VC stating that the intended identity is the subject in control of some DID, e.g., *did:test:alice-long-term-did*. This credential is issued by some institution that is trusted by the communication partners of the Prover.
- Linkage to the Communication DID - a self-asserted VC issued by the Prover themselves using the keys in control of the long-term DID *did:test:alice-long-term-did*. The communication DID of the Prover is contained in this credential‘s payload which links the intended identity to the communication DID.

A similar approach for preventing MITM attacks when using DIDComm Messaging in combination with Peer-DIDs has been presented in [13], however, due to a poorly described

[13]DIDComm Guidebook - MITM attacks - `https://didcomm.org/book/v2/mitm`

[14]Should we address MITM detection at all within DIDComm-messaging? - `https://github.com/decentralized-identity/didcomm-messaging/issues/41`

solution, we could not infer the exact details about the exchanged credentials.

In the context of an Access Control System (ACS), the user presents their credentials to the ACS in order to access the requested resources. As part of the authorization, the ACS can use the presented approach and request a proof from the user that links their identity to the DID they are currently using for communication. If a MITM attack is indeed present, the ACS will detect it prior to the sharing of the resources, thus, the server resources remain successfully protected. However, the detection of the MITM attack requires the presentation of the user credentials. Therefore, if the adversary‘s goal is to steal user credentials, which is a completely rational attack purpose, the attacker can be successful. For example, a project by the German ministry for developing an SSI-based smartphone application for a digital identity wallet[15] has been criticized [124] to be vulnerable against MITM attacks allowing an adversary to steal user credentials.

As a conclusion, the prevention of MITM attacks when using the DIDComm Messaging protocol is a complicated topic which requires further research in both defining desired security properties (what should be protected) and designing and implementing concrete prevention schemes. Furthermore, even if the attack should be prevented on the application level, at least some description of the problem and possibly some recipe or best practices on how to implement such prevention, should be added to the DIDComm Messaging specification.

4.5 Exchange of Credentials

The VC Data Model specification defines how VCs should be packed into a VP so that a Verifier can cryptographically be ensured that the credentials belong to the entity with which they are currently communicating. However, for the complete exchange of credentials between a Verifier and a Holder, two more important details should be covered:

- **Requesting Credentials:** In some scenarios, the Verifier needs to specify what types of credentials are expected from the Holder. Especially in a more complex system with many access control rules, the required credentials might depend on the requested resource and even on the requested operation on this resource. To solve such use cases, a mechanism for the Verifiers to describe requirements on expected credentials is needed. In many cases, a VP also needs to include a cryptographic challenge chosen by the Verifier to prevent Replay-Attacks. Usually the Verifier sends this challenge to the Holder in a separate message prior to the exchange of credentials. To minimize the number of exchanged messages, it makes sense to

[15] German ID Wallet - `https://www.bundesregierung.de/breg-de/suche/e-id-1962112`

combine the challenge and the description of required credentials in a single message, often denoted as a Verifiable Presentation Request (VPR).

- **Interoperable and extensible protocol for exchange of credentials:** In addition to the mechanisms and formats of the VPRs and VPs, an interoperable protocol defining the messages that are sent by the Verifier and Holder is required. In this way, the two parties can parse, decode and understand the semantics of the received messages. Furthermore, such a protocol should be independent on the specific mechanisms used for creating the VPRs and VPs.

4.5.1 DIF Presentation Exchange

The DIF Presentation Exchange [19] is a specification which aims to standardize the way Verifiers describe requirements on expected credentials and Holders describe submissions of credentials which satisfy these requirements. The specification aims to be both credential format and transport envelope agnostic, i.e., Presentation Exchange can be used when exchanging different flavors of credentials using different proof types (see section 4.3) over various transport mechanisms, such as DIDComm, OpenID Connect and others. The specification defines the following objects:

- **Presentation Definition** - used by the Verifier to describe requirements on the credentials they are expecting. The Presentation Definition is the top-level object which consists of one or multiple Input Descriptors Objects each describing a single credential. The Presentation Definition is satisfied when all of the included Input Descriptors are satisfied (unless otherwise specified by a Feature which will be introduced below).
- **Input Descriptor Object** - description of the information (the payload of the credential) required by a Verifier. An Input Descriptor Object must have a unique identifier and may contain the following properties:
  - **name** - human-friendly name describing the requirement.
  - **purpose** - describes why the information or credential is being requested.
  - **format** - describes accepted formats for the credential. Should be an object containing one or multiple properties matching the so-called registered *Claim Format Designations*, such as jwt_vc, jwt_vp and ldp_vc.
  - **constraints** - a list of objects consisting of multiple fields, such as *path*, *filter* and *required*. The constraints define specific requirements on the credential, such as specific values stored in the credential‘s payload.

- **Presentation Submission** - integrated in the message containing a Verifiable Presentation and used by the Holder to express how the presentation fulfills the requirements defined in a previously received Presentation Definition. The Presentation Submission object references a Presentation Definition by its identifier and must include a *descriptor_map* property whose value is a list of *Input Descriptor Mapping Objects*. An Input Descriptor Mapping Object references an Input Descriptor object from the Presentation Definition and provides a relative path (in the form of a JSONPath[16]) to the submitted credential which satisfies this descriptor.

- **Features** - extensions to the Presentation Definition enabling the Verifiers to express more complex requirements, such as combination of credentials or usage of privacy-protecting mechanisms when providing the credentials, e.g., Zero-Knowledge Proofs with predicates. For simplicity, only the **Submission Requirement Feature** is explained in this work.

  The Submission Requirement Feature enables Verifiers to define combinations of Input Descriptors. When using this Feature, each Input Descriptor must have the group property, which defines the group to which this descriptor belongs. The Presentation Definition object must also include the *submission_requirements* property which includes one or multiple objects defining combinations of descriptors, e.g., "satisfy 1 of the Input Descriptors of group A, at least 2 of the Descriptors of group B and all of the Descriptors of group C".

DIF Presentation Exchange has been integrated in multiple SSI Ecosystems, including Hyperledger Aries, and has a high probability of establishing itself as the de-facto standard for exchange of credentials [26]. Therefore, it is beneficial for emerging systems, including the Access Control System we are building, to support it.

DIF Presentation Exchange is well-suited for machine processing, such as automatic selection of credentials that satisfy the defined requirements and verification whether the provided credentials satisfy the required ones. However, due to the very long and complex structure of the used JSON Objects (for examples, see the specification [19]), the Presentation Definitions and Submissions are not well-suited for human-readability, which is an important property for Access Control Rules, as they are often written and read by End-Users. Therefore, in this work we have decided to analyze and use other technologies (Web Access Control 3.2.2 in combination with RDF Data Shapes 2.5.3, 2.5.4) for the modelling of the Access Control Logic and only use DIF Presentation Exchange on the level of credential exchange.

[16]JSONPath (a syntax used to reference specific parts of a JSON object) Python implementation - `https://github.com/kennknowles/python-jsonpath-rw`

4.5.2 HL Aries Present Proof Protocol

The Hyperledger Aries Present Proof Protocol 2.0 [63] is a protocol for a general exchange of Verifiable Credentials between a Verifier and a Holder that is independent on the specific formats of the exchanged Verifiable Presentation Request and Verifiable Presentation. The protocol defines the messages supporting the exchange of Verifiable Credentials, their structure and how the messages should be handled. The protocol defines the two roles - **Prover** and **Verifier**, and the following messages:

- **Propose-Presentation:** An optional message sent by the Prover to the Verifier either prior to any communication in order to initiate a credential presentation process or as a counter-proposal in response to a received Request-Presentation.
- **Request-Presentation:** Sent by the Verifier to the Prover. The message contains a Verifiable Presentation Request in one or multiple formats which are also specified as part of the message. The structure of the message is presented in Listing 1 contained in the appendix. The *formats* list of the Request-Presentation contains an entry for each, so-called *attachment*, in the *request_presentations attach* list, specifying the format of the Verifiable Presentation Request described by this attachment.
- **Presentation:** Sent by the Prover to the Verifier in a response to a Request-Presentation. The message contains a Verifiable Presentation in a specific format. The structure of the message is shown in Listing 2 contained in the appendix.
- **Acknowledge:** The sender of this message confirms to the receiver that a specific other message was received.
- **Problem-Report:** The sender reports an occurred error to the receiver.

4.6 Summary

In this chapter, the current state of Self-Sovereign Identity was presented by discussing the challenges on the road towards the successful integration of SSI on the Web and analyzing some important underlying SSI technologies. Section 4.1 identified and summarized the major SSI challenges, the most notable of which are the interoperability between vertically-competing and horizontally-dependent SSI technologies and the high complexity of the SSI paradigm. In section 4.2, multiple DID Methods (Peer-DIDs 4.2.1, Ethr-DIDs 4.2.2, Indy-DIDs 4.2.3, KILT-DIDs 4.2.4 and Web-DIDs 4.2.5) were introduced and the DIF Universal Resolver framework 4.2.6 was suggested as a solution for the interoperability problem between them. After that, section 4.3 summarized 4 flavors of Verifiable Credentials - JSON-LD Credentials with LD-Signatures, JSON-LD Credentials with BBS+ Signatures, JSON or JSON-LD Credentials expressed as JSON Web Tokens (JWTs), and the JSON Anonymous Credentials, and presented the differences between them in the various stages of a credential‘s lifespan. Multiple approaches for achieving a secure communication on the Web, including the approach of using the DIDComm Messaging Protocol in combination with Peer-DIDs, were discussed in section 4.4. Finally, section 4.5 presented the DIF Presentation Exchange 4.5.1 - a standardized way for the Verifiers to express requirements on the credentials they are expecting from a Holder, and the HL Aries Present Proof Protocol 4.5.2 which defines the types and formats of messages sent between a Verifier and a Holder during a generic exchange of Verifiable Credentials.

SSI-Interoperable Access Control System

The Interoperable Access Control System (ACS) based on Self-Sovereign Identities presented in this work is designed with the following goals in mind:

- **Authorization based on Self-Sovereign Identities:** Users of the system prove claims about themselves by using their Decentralized Identifiers (DIDs) and the Verifiable Credentials (VCs) issued to them. The Access Control System supports procedures for exchange of credentials, verifies the received user credentials, and makes authorization decisions based on their validity, issuers, payloads and the internally defined access control policies.
- **Interoperability of DID Methods and VCs:** The ACS should support multiple DID Methods and flavors of VCs in an extensible way, i.e., the design of the ACS should allow for an incremental integration of additional DID Methods and types of VCs.
- **Interoperability with existing SSI Agent implementations:** For existing SSI Agent implementations to be interoperable with the ACS a relatively low overhead for updating the functionality of the agents should be required. To achieve this, the ACS should use well-established and widely-supported technologies and protocols for the communication with the users and other procedures, such as exchange of credentials.
- **User-understandable Access Control Logic:** The access control logic should be defined using *user-understandable* authorization rules describing specific access modes (such as read and write operations) to specific resources. The authorization rules should be based on a well-established design but should also be enriched with the additional functionality to specify required credentials and describe required properties of these credentials, such as sets of accepted issuers or specific values in the credentials' payloads. The modelling of required credentials should be done using semantic technologies conforming to the Linked Data Principles.

As a Proof-of-Concept (PoC) for the viability of the system, a prototype of the designed Access Control System is implemented and evaluated with regard to the performance of the authorization procedure and the achieved interoperability aspects. This chapter presents the detailed design of the *Access Control System (ACS)* and the implementation of its *PoC-Prototype*.

It is important to mention that due to time constraints, some parts of the implementation are incomplete or do not fully conform to the defined design. For example, the system design states that the *DIF Presentation Exchange* protocol (see section 4.5.1) should be used in addition to the newly-defined credentials exchange protocol, as this would improve the interoperability of the Access Control System with already existing SSI Agent implementations. However, the PoC-Prototype‘s module responsible for the DIF Presentation Exchange implementation is a non-functional dummy module which only show-cases how the protocol could be integrated in the ACS and is currently marked as future work. Another example for the incomplete implementation of the system would be that for simplicity and a faster prototype development, local files are used for persistent storage instead of creating and deploying an actual database.

5.1 Overall System Design

The Access Control System (ACS) is designed as a Microservice architecture consisting of independent components deployed as Docker containers communicating with each other over well-defined APIs. Each service implements a specific functionality, such as resolving DIDs of a specific DID Method or verifying a specific type of VCs. The Microservice architecture allows for an incremental extensibility by simply adding additional services to the design which introduces minimal changes to the existing services. Furthermore, substituting a specific functionality, e.g., updating the communication protocol, is as simple as updating or exchanging the component responsible for this functionality with a new one. As long as the new component retains to the previously defined APIs, the other parts of the system should only require minor-overhead updates or none at all. This independent evolvability of the components is crucial to the system design, as due to the rapid development within the SSI Ecosystem, such updates of the core SSI technologies are to be expected.

Another advantage of the Microservice architecture is that it allows for different programming languages to be used in the different services. Due to the decentralized nature of the SSI paradigm, different tools and projects are provided by different organizations and are written in different languages. All of the components, that we have implemented on our own, are written in the Python programming language, however, other open-source parts of the ACS which have been contributed by the SSI Communities are written in

other languages. For example, the Universal DID Resolver is a Java application which connects with many other DID drivers - the resolver for Ethr-DIDs [1] is written in the programming language Typescript, while the resolver for Indy-DIDs [2] in Rust.

The complete design of the ACS is presented in figure 5.1. The interoperable SSI-based ACS consists of three main packages:

- **DID-Interoperability** - resolves different types of DIDs to their DID Documents. Due to the standardized data model of the DID Documents, the ability to resolve the different DIDs is sufficient to achieve DID-Interoperability on the level required by the ACS. The DID-Interoperability package deploys a local instance of the DIF Universal Resolver 4.2.6 and configures the DID Drivers for the supported DID Methods.
- **Communication-over-DIDs** - handles the communication with the End-Users which is currently based on the DIDComm Messaging protocol and Peer-DIDs as presented in section 4.4.4. The package consists of the single component *DID Communication API* which is responsible for receiving DIDComm Messages from the End-Users, decrypting and authenticating them, and forwarding them to the configured message handler of the ACS. All of the communication between the End-Users and the ACS is forwarded through the DID Communication API.
- **Access Decision Point (ADP)** - the most sophisticated package which handles the entire logic of the ACS. ADP manages the Access Control Rules (ACRs), handles the received messages, creates Presentation Requests, verifies Verifiable Presentations, validates provided credentials against the defined ACRs, makes authorization decisions and instructs the DID Communication API component on how to respond to the End-Users.

Outside the scope of the ACS, a single End-User is depicted. One of the defined goals is to not require an overly-complex functionality on the client-side, however, the clients of the ACS must be aware of how to use and communicate with the ACS, e.g., how to pack DIDComm Messages in HTTP Requests. Therefore, an additional **SSI-Client** component on the client-side is shown. We differentiate between the terms End-User (the entity trying to authorize itself - a person, thing or an organization) and SSI-Client (a user-agent that creates and sends DIDComm Messages packed in HTTP Requests to the ACS on behalf of the End-User).

[1] `https://github.com/decentralized-identity/ethr-did-resolver` - Ethr-DID Resolver
[2] `https://github.com/IDunion/indy-did-resolver` - Indy-DID Resolver

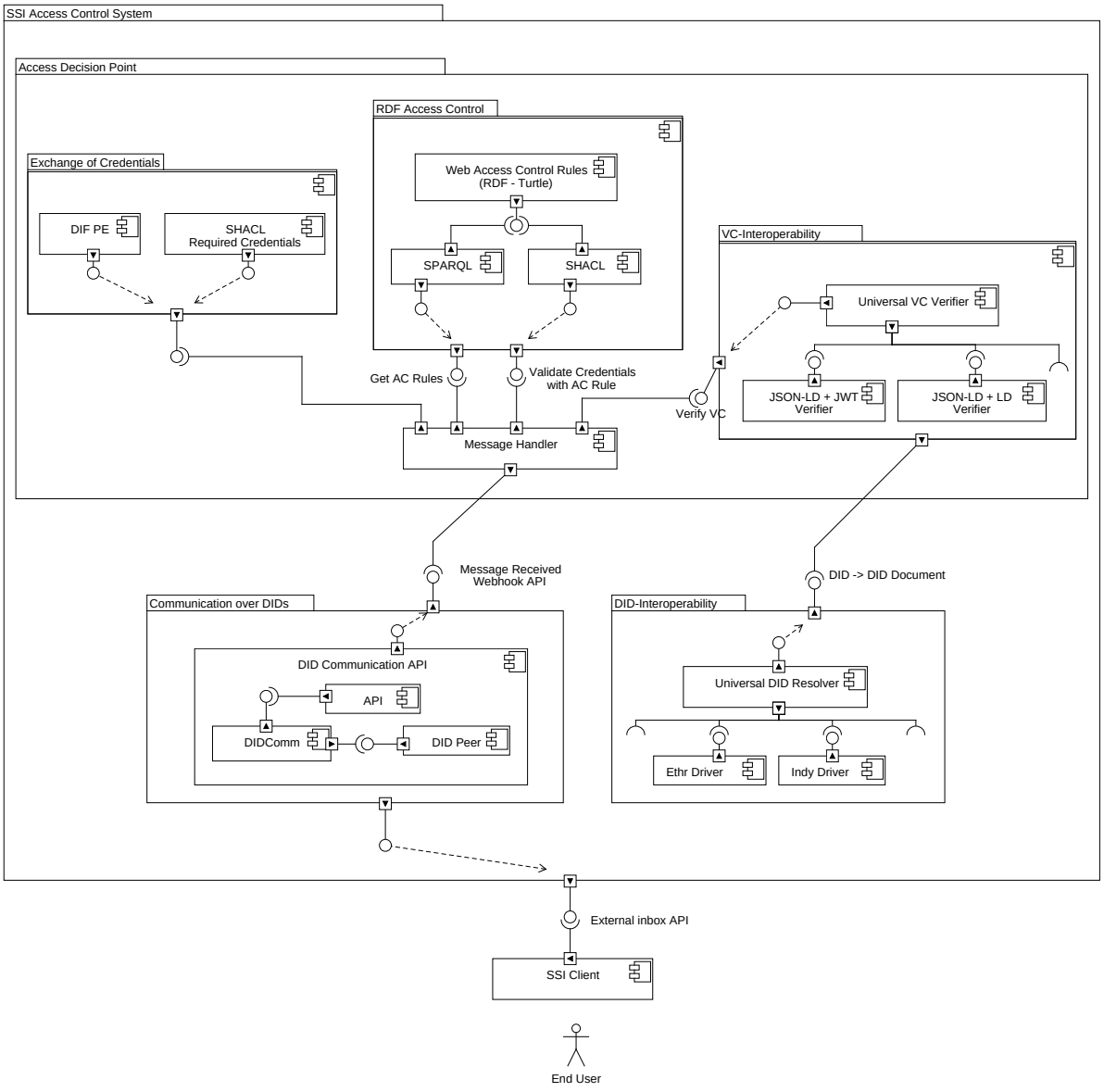


Figure 5.1: Complete design of the ACS

5.2 DID Communication API

The DID Communication API service (see figure 5.2), also referred to as *DID-Comm-API*, is responsible for the secure communication between the ACS and the SSI-Clients. The communication is secured using the DIDComm Messaging Protocol [30] and static Peer-DIDs 4.2.1 as suggested in section 4.4.4. The DID-Comm-API receives user requests containing DIDComm Messages, decrypts and authenticates them, notifies the configured webhook (the internal API Endpoint of the ADP) about the received messages, and responds to the user requests as instructed by the webhook‘s response. In this way, all communication between the ACS and the SSI-Clients is forwarded through and secured by the DID-Comm-API. This extraction of the communication in a separate service hides the unnecessary details (the management of Peer-DIDs, the encryption, decryption and verification of DIDComm messages, and the underlying transport protocol used by the DIDComm protocol) from the upper layer responsible for the access control logic. Currently, DID-Comm-API uses the HTTP protocol as the underlying transport for the DIDComm messages.

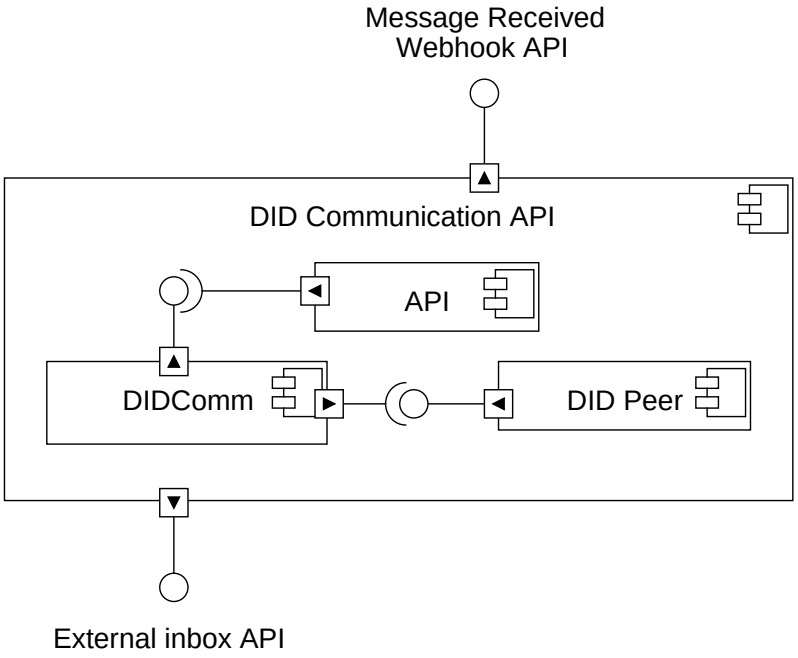


Figure 5.2: DID Communication API

We have designed and implemented two versions of this service, which are referred to as DID-Comm-API-v1 and DID-Comm-API-v2. In the first version, the HTTP protocol was only used as a transport for the DIDComm messages, i.e., the SSI-Client would create a DIDComm Message, pack it into an HTTP Request and send the HTTP Request to the

inbox API of DID-Comm-API-v1. DID-Comm-API-v1 unpacks the DIDComm message, decrypts and authenticates it, and immediately sends an HTTP Response to the SSI-Client. This HTTP Response only acknowledges the received DIDComm Message and does not carry any semantic meaning about the processing of the Client's request. DID-Comm-API-v1 then notifies ADP of the received message. Once ADP responds to the DID-Comm-API-v1, it creates a new DIDComm Message and packs it in another HTTP Request which is then sent to the service endpoint of the SSI-Client.

To facilitate this exchange of DIDComm Messages between the two service endpoints of the ACS and the SSI-Client, DID-Comm-API-v1 provides two API endpoints - */did-comm/inbox/* and */did-comm/init-connection/*. Prior to using the inbox API, SSI-Clients first need to use the init-connection API in order to establish a "connection" with DID-Comm-API-v1. A connection consists of a fresh Peer-DID generated by the SSI-Client, a fresh Peer-DID generated by DID-Comm-API-v1 and the service endpoint of the SSI-Client. It is also important to mention, that DID-Comm-API-v1 uses a fresh Peer-DID for each new SSI-Client.

DID-Comm-API-v1 is entirely designed for P2P-oriented communication which makes sense for many use cases in the SSI ecosystem. However, there are multiple drawbacks of this version:

- **Complexity of the Design:** Establishing and managing this connection-oriented view introduces a lot of complexity in the otherwise simple DID Communication API service.
- **Complexity of the Implementation:** The implementation of this version is also unnecessary complex as it requires managing multiple connections and asynchronous handling of user requests, even after sending HTTP responses.
- **Man-in-the-Middle Attacks:** A MITM attack is possible during the explicit exchange of Peer-DIDs between the SSI-Client and the ACS.
- **Not Web-oriented:** This version of the DID Communication API service is well suited for Peer-to-Peer communications, however, the ACS should obviously play the role of a server and can benefit from using the semantics of the HTTP protocol. Specifically, the status codes of HTTP responses can be utilized by the ACS.
- **Complex client-side Functionality:** In this version, the SSI-Clients must also provide and manage a service endpoint and must wait and receive DIDComm Messages on it. As this cannot be evaluated as a low-overhead update for existing SSI Agent implementations, this contradicts one of the defined goals of the ACS.

Due to the presented drawbacks, a second version of the DID Communication API was

designed and implemented. DID-Comm-API-v2 fulfils the same functionality as the first version, but is more aligned with the server role of the ACS and utilizes HTTP semantics.

On startup, DID-Comm-API-v2 generates and outputs a single Peer-DID, further denoted as the *Server-DID*. The Server-DID (which also contains the DIDComm Service Endpoint of the DID-Comm-API) is published by the ACS and used by all SSI-Clients to encrypt their DIDComm Messages. How exactly the Server-DID is disseminated on the Web depends on the use-cases of the server and is considered out-of-scope for our work. DID-Comm-API-v2 provides only a single */didcomm/inbox/* HTTP API on which SSI-Clients send HTTP Requests containing DIDComm Messages. DID-Comm-API-v2 decrypts and authenticates the received DIDComm Messages and notifies ADP of the received user requests. Instead of directly responding to the SSI-Client, DID-Comm-API-v2 waits until ADP handles the user request. Once, ADP responds to DID-Comm-API with instructions on how to respond to the SSI-Client, DID-Comm-API-v2 creates a DIDComm Message according to the received instructions and packs it into an HTTP Response which is then sent back to the SSI-Client. ADP has also been updated to additionally specify the HTTP status code of the HTTP Response sent back to the SSI-Client. Using this new version of DID-Comm-API has the following advantages:

- **Simplified client-side Functionality:** SSI-Clients are not required to establish a connection to the ACS prior to sending their requests. Furthermore, they do not need to provide and manage an HTTP service endpoint.
- **Less Overhead:** Firstly, the ACS does not need to store and manage a context for each connection. Secondly, the additional connection establishment prior to the user requests is avoided. DID-Comm-API-v2 is, therefore, more efficient both in server resources and in End-to-End latency.
- **Web-oriented:** DID-Comm-API-v2 is more aligned with the server role of the ACS and can utilize the semantics of the HTTP protocol. For example, if the SSI-Client needs to provide a credential as part of the authorization process, it makes sense to pack the Presentation Request in an HTTP Response with status code HTTP 401 Unauthorized, which requests authorization from the client. If the credentials provided from the SSI-Client do not satisfy the required credentials, then the ACS can respond with an HTTP Response with status code 403 Forbidden, which semantically means that the SSI-Client has authenticated himself successfully (the client‘s credentials have been verified successfully), however, the provided credentials are insufficient for accessing the requested resource.

Currently, the DID-Comm-API only supports communication on top of Peer-DIDs, i.e., the SSI-Clients do not use their original DIDs for securing the communication with the

ACS, but instead they also generate fresh Peer-DIDs and use them. The advantages of this approach are presented in section 4.4.4, however, as it should usually be up to the End-Users to decide which DIDs to use for the communication, the DID-Comm-API should be updated (as a future work) to also support other DID Methods for the communication. This is possible by connecting DID-Comm-API to the Universal DID Resolver component which we evaluate as a low-overhead future task due to the flexible Microservice design of the ACS.

Finally, it is important to note that in the current system design no concrete solution for prevention of Man-in-the-Middle (MITM) attacks has been integrated. As explained in section 4.4.5, MITM prevention in the context of the DIDComm Messaging protocol is a complicated topic that needs a careful definition of desired security guarantees and design of concrete prevention schemes which can also be application-specific. For example, MITM prevention is dependent on how the Server-DID of the ACS is published on the Web or how a client of a resource server is forwarded to the ACS in order to perform the authorization. Furthermore, MITM prevention might also be dependent on the SSI solutions of the user. In any case, designing and implementing concrete MITM prevention techniques remains an open problem and should be further researched as a future work.

5.3 Univeral DID Resolver

To provide a solution to the DID-Interoperability problem, the ACS deploys a local instance of the DIF Universal Resolver (see section 4.2.6). The architecture of the locally deployed instance of the Resolver is depicted in figure 5.3. In the scope of this work, we have used Ethr-DIDs (sec section 4.2.2) on the Ropsten test network and Indy DIDs (see section 4.2.3) on a local instance of the VON Network. Therefore, two drivers have been deployed - the Ethr-DID driver and the Indy-DID driver. Because we used a local VON Network, the Indy-DID-driver was updated [3] and configured to connect to our local VON-network. Otherwise, the deploying of the drivers is as simple as running a Docker container. With the Universal DID Resolver, all parts of the ACS can resolve different types of DIDs. Furthermore, the architecture can be extended by simply deploying additional drivers for other DID Methods. Therefore, the goal of Interoperability with regard to DIDs is considered to be theoretically fulfilled, however, the performance of the Universal Resolver and the differences between the resolution times of the different DID Methods will be further analyzed in the Evaluation chapter.

[3] `https://github.com/IDunion/indy-did-resolver/issues/17` - How to connect Indy DID Driver to a local network

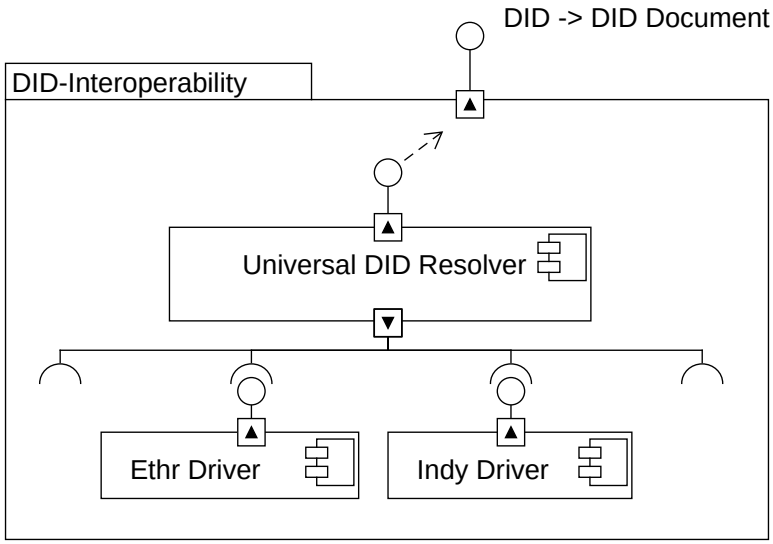


Figure 5.3: Universal Resolver

5.4 Access Decision Point

The Access Decision Point (ADP) is the main package of the ACS containing the services responsible for the definition and enforcement of the access control logic. As shown in figure 5.4, ADP consists of the following services:

- **Message Handler** - handling of user requests: The ADP's Message Handler component provides a single HTTP API, used as a webhook from the DID-Comm-API. In this way, DID-Comm-API notifies ADP whenever a new user request (in the form of a DIDComm Message) is received. ADP handles the received message and instructs DID-Comm-API on how to respond to the received request. The formats of the notification that a new user request was received and the format of the Message Handler's response are presented in the listings 3 and 4 (contained in the appendix) respectively. Currently, only user communication in the form of DIDComm Messages over the HTTP protocol is supported, however, the API's structure allows for extensibility by defining further request types. During the handling of the received user request, the Message Handler connects to the other services of the ADP for procedures such as exchange and verification of credentials, and making authorization decisions.

- **RDF Access Control** - defining the access control logic: The Access Control service of the ADP package manages the access control file which stores the authorization rules for all resources and defines the types of required credentials and their descriptions. The access control is defined using authorization rules based on

Solid's Web Access Control (see section 3.2.2) but enriched with further semantics for defining the required credentials that the users need to provide. A simple example access control file containing a single authorization rule and a single required credential is given in listing 5.1. It defines the access logic that users wanting to read the resource identified by URI `http://aifb.example.org/resources/r1` need to provide a specific type of a credential (`https://example.org/examples#UniversityDegreeCredential`) issued by one of the specified issuers and including the specific value "Bachelor of Science" as the degree name.

As the authorization rules and the descriptions of the required credentials are RDF documents serialized in Turtle format, ADP also implements the functionality for parsing the authorization rules using SPARQL (see section 2.5.2) and validating provided credentials against the defined descriptions of the required credentials using SHACL (see section 2.5.4). The reasons why we have chosen to model the required credentials using SHACL are discussed in the comparison between SPARQL, SHACL and ShEx in the Evaluation chapter (see section 6.2.4).

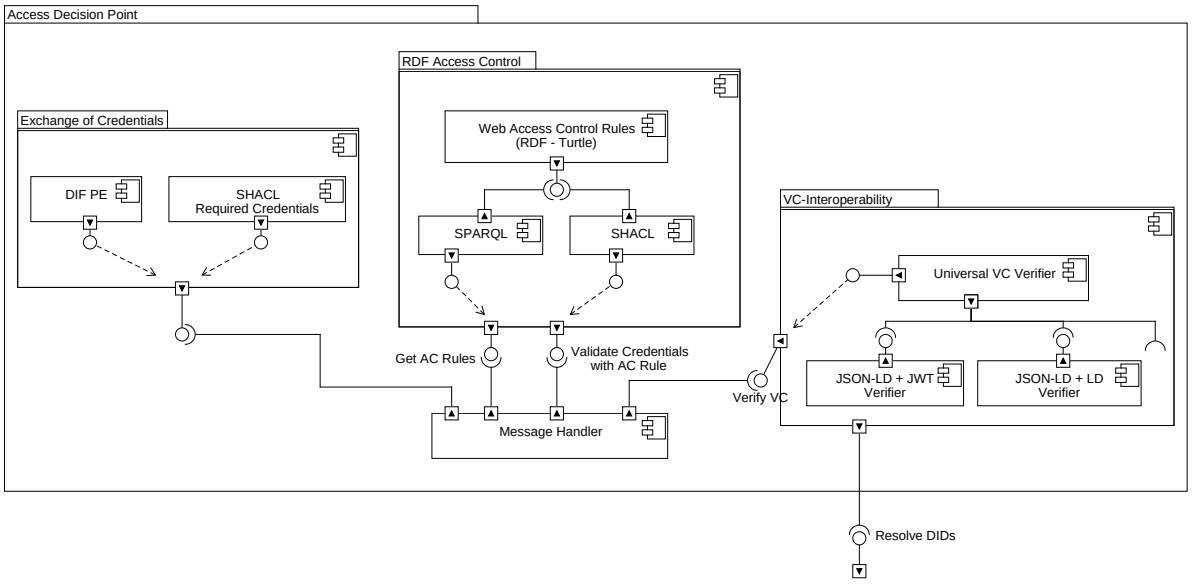


Figure 5.4: Design of Access Decision Point (ADP)

- **Exchange of Credentials** - depending on the requested resource and defined authorization rules, an SSI-Client might be required to provide a Verifiable Presentation with some credentials in order to be given access to the resource. In such cases, the Exchange of Credentials service of ADP is responsible for requesting the credentials from the SSI-Client (by creating a Presentation Request and instructing DID-Comm-API to send it) and receiving and parsing the Verifiable Presentation, once the SSI-Client provides it. ADP currently uses the Hyperledger Aries Present-Proof protocol (see section 4.5.2) for building and parsing the messages requesting

and presenting credentials. In the Present-Proof protocol, the verifier (ADP in our case) can specify the required credentials in different formats. We have defined and used our own format [4] [5] based on the SHACL descriptions of required credentials, in which the presentation request contains the SHACL definitions of the required credentials. Additionally, the system design also specifies a translation of the SHACL required credentials descriptions to the currently widely-adopted DIF Presentation Exchange protocol (see section 4.5.1) for better interoperability with existing SSI agent implementations. However, it is worth mentioning, that due to time constraints this translation has not been implemented and is regarded as future work.

- **VC-Interoperability** - verification of Verifiable Presentations (VPs) and Verifiable Credentials (VCs) in different formats. ADP is also responsible for verifying the provided VPs and VCs by the SSI-Client. For interoperability with different types of credentials (see section 4.3), ADP finds out the type of the VP or VC at hand and connects to the appropriate Verifier. The system design considers deploying multiple verification services, each responsible for verification of certain type of credentials. In the scope of this work, due to time constraints, only a single verifier has been implemented - verifier for JSON Linked-Data Credentials with JSON Web Token Proofs. The JSON-LD + JWT Proofs Verifier is further discussed below as a separate component of the ACS.

A note on the implementation of the PoC-Prototype: each of the listed ADP services is defined as a separate Microservice and thus an independent component in the design architecture. In the PoC-Prototype, however, the entire ADP has been implemented as a single Python Flask project and is deployed as a single Docker container. To practically separate and isolate the ADP services, they have been implemented as separate Python packages in the ADP project. Only the verifier of JSON-LD+JWT credentials is an exception to this, as it has been implemented as a separate Python project and deployed as a completely independent Docker container.

[4] Presentation Request SHACL Attachment Format - `https://uwmbv.solid.aifb.kit.edu/ssi-acs/didcomm/attachments/required-credentials/SHACL/presentation-request`

[5] Presentation SHACL Attachment Format - `https://uwmbv.solid.aifb.kit.edu/ssi-acs/didcomm/attachments/required-credentials/SHACL/presentation`

```
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix acl:    <http://www.w3.org/ns/auth/acl#> .
@prefix aifb:   <http://aifb.example.org/> .
@prefix sh:     <http://www.w3.org/ns/shacl#> .
@prefix schema: <http://schema.org/> .
@prefix cred:   <https://www.w3.org/2018/credentials#> .
@prefix ex:     <https://example.org/examples#> .

aifb:resource_requires_degree_science
    a                       acl:Authorization ;
    acl:accessTo            <http://aifb.example.org/resources/r1> ;
    acl:agent               acl:AuthenticatedAgent ;
    acl:mode                acl:Read ;

    # Extension to WAC: required credential property
    aifb:requiredCredential aifb:BachelorScienceDegreeCredentialShape .

aifb:BachelorScienceDegreeCredentialShape
    a              sh:NodeShape ;
    sh:targetClass cred:VerifiableCredential ;

    # Credential of specific type:
    sh:class       ex:UniversityDegreeCredential ;

    # DIDs of Accepted Issuers:
    sh:property    [
        sh:path cred:issuer ;
        sh:in   ( <did:ethr:0x3:0x032..5d7>
                  <did:indy:local:YY8..6tZ> ) ;
    ] ;

    # Exact value stored in the credential:
    sh:property    [
        sh:path     ( cred:credentialSubject ex:degree schema:name ) ;
        sh:hasValue "Bachelor of Science"^^rdf:HTML ;
    ] .
```

Listing 5.1: Example Access Control File

5.5 JSON-LD with JWT Proofs Verifier

The JSON-LD+JWT Verifier (see VC-Interoperability package in figure 5.4) service (further denotes simply as JWT-VC-Verifier) is responsible for verifying Verifiable Credentials and Verifiable Presentations expressed as JSON Web Tokens (JWTs) and to translate the JWT-specific payloads to the W3C VC Data Model. Because JWTs use some standardized fields to express attributes of the token, such as issuers and expiration dates, which are also part of the data model of a VC, the W3C Data Model [105] specifies how VCs and VPs should be expressed as JWTs and how the payloads of these JWTs should be translated back to the original W3C Data Model. To provide a unified data model for the credentials across the entire ACS, the JWT-VC-Verifier implements this translation.

The JWT-VC-Verifier provides two HTTP APIs - */verify/vc* and */verify/vp* - for verification of VC and VPs. The response formats of the APIs are given in listings 5 and 6 (contained in the appendix), and contain a verification result, the original payload of the JWT, the DID of the issuer of the credential (or holder in case of a VP), and the payload of the credential or presentation translated to the W3C data format. As part of the verification, the JWT-VC-Verifier uses the locally deployed Universal Resolver service to resolve the DID specified as the issuer of the JWT.

5.6 The Complete Authorization Process

In this section, an overview of the complete authorization procedure is given, i.e., all steps overtaken by both the components of the ACS and by the SSI-Client. The authorization process is presented both graphically as a sequence diagram (see figures 5.5 and 5.6) and textually as a sequence of sub-procedures:

1. ACS Initialization:
   (a) The Resource Owner defines the Access Control by supplying ADP with the access control file containing the Authorization Rules and SHACL descriptions of required credentials.
   (b) DID-Comm-API generates and publishes the Peer-DID of the ACS
2. SSI-Client Initialization:
   (a) Load the Holder‘s credentials and parse them as RDF Graphs (executed during initialization to improve efficiency as explained in the Evaluation chapter)
   (b) Generate a Peer-DID

3. SSI-Client creates an *Initial Request* which contains the URI of the requested resource.

4. SSI-Client packs the Initial Request into a DIDComm Message. The DIDComm Message is encrypted using the Peer-DID of the ACS and authenticated using the Peer-DID of the SSI-Client.

5. SSI-Client sends an HTTP Request on the inbox API of DID-Comm-API. The DIDComm Message is contained in the payload of the HTTP Request.

6. DID-Comm-API parses the HTTP Request, decrypts the DIDComm Message using the ACS's Peer-DID and stores the SSI-Client's Peer-DID (which is included in the DIDComm Message) for later use.

7. DID-Comm-API notifies the configured webhook (ADP's API) for the received request and waits for instructions on how to respond to the SSI-Client. This notification contains the SSI-Client's Peer-DID, the type of received HTTP Request (GET/POST/PUT/..) and the decrypted payload and metadata of the received DIDComm Message.

8. ADP handles the received DIDComm Message as an Initial Request (as specified by the type of the DIDComm Message):

   (a) Load required credentials - ADP queries the RDF Access Control File using SPARQL and finds out the SHACL descriptions of required credentials for the requested resource and operation.

   (b) Create Presentation Request - using the Present-Proof Protocol 4.5.2, ADP creates a message containing a Verifiable Presentation Request (VPR). Currently, only a single VPR format based on the SHACL descriptions is used, however, as future work, DIF Presentation Exchange 4.5.1 should also be supported. A cryptographic challenge (nonce and domain) is also included in the VPR.

   (c) Respond to DID-Comm-API: ADP responds to the DID-Comm-API's webhook request by instructing it to pack the message containing the VPRs into a DIDComm Message and to respond to the SSI-Client's request using an HTTP Response of status code 401 - Unauthorized. The DIDComm Message is included in the HTTP Response payload.

9. DID-Comm-API responds to the SSI-Client as instructed by ADP. This includes a single DIDComm Pack-Message operation using the Peer-DIDs of the ACS and SSI-Client.

10. SSI-Client handles the ACS‘s response:
    (a) Unpack the DIDComm Message included in the HTTP Response
    (b) Parse the VPR
    (c) Select the credentials from the local wallet that satisfy the VPR. This includes executing a SHACL validation for each locally stored credential against each required credential SHACL description.
    (d) Ask End-User for consent. In a real scenario, the SSI-Client should always query the End-User for approval when providing any credentials to an external system. This step is only mentioned for the sake of theoretical completeness and is otherwise ignored in our prototype implementation of an SSI-Client.
    (e) Create a Verifiable Presentation (VP) containing the selected credentials and the requested cryptographic challenge (nonce and domain). The cryptographic proofs of the VP and the included credentials are all represented as JSON Web Tokens, as currently this is the only supported format in our prototype.
    (f) Pack the VP in a so-called *Secondary User Request* - another DIDComm Message also packed in an HTTP Request. The Secondary User Request is then sent on the inbox API of DID-Comm-API.
11. DID-Comm-API parses the received HTTP Request, decrypts the DIDComm Message using the ACS‘s Peer-DID and stores the SSI-Client‘s Peer-DID (which is included in the DIDComm Message) for later use. DID-Comm-API once again notifies ADP and waits for instructions on how to respond to the SSI-Client.
12. ADP parses the VP and uses the appropriate VC-Verifiers to verify the VP and the included VCs. Currently, only JWT-based credentials and presentations are supported. ADP calls the *verify-vp* API of the JWT-VC-Verifier.
13. JWT-VC-Verifier decodes the JWT representing the VP and parses the Holder‘s DID.
14. JWT-VC-Verifier sends a resolve-DID request for the Holder‘s DID to the Universal Resolver
15. The Universal Resolver finds out the type of the DID and connects to the appropriate DID Driver which resolves the DID and returns a DID Resolution object. The Universal Resolver responds to JWT-VC-Verifier with this DID Resolution object.
16. JWT-VC-Verifier parses the DID Resolution object which contains the resolved DID Document (assuming successful resolution). Using the cryptographic materials in

the DID Document, the JWT-VC-Verifier verifies the VP. Upon successful verification, JWT-VC-Verier also translates the JWT payload to the W3C VC Data Model format for Verifiable Presentations. JWT-VC-Verifier sends a response to ADP containing the verification result, a failure reason (in case of unsuccessful verification), the JWT payload and the VP translated in W3C Data Format.

17. ADP parses the response of the JWT-VC-Verifier. ADP also checks whether the requested cryptographic challenge is included in the VP.

18. ADP uses the JWT-VC-Verifier (as the only VC Verifier) to verify each of the credentials included in the VP. The process is similar to the verification of a VP. The JWT-based credentials are also translated to the W3C VC Data Format. It is important to mention, that a single DID resolution is required for the issuer of each credential included in the VP.

19. Following a successful verification of the VP and the included credentials, ADP validates whether the provided credentials satisfy the required credentials for the requested resource: ADP checks whether each of the SHACL descriptions representing a required credential is satisfied by at least one of the provided credentials.

20. In the case of a successful verification and validation of the credentials, ADP grants the SSI-Client access to the requested resource. In the prototype implementation, the resources are an abstraction. Instead of granting an actual access to a resource, the ACS simply responds to the SSI-Client with a message representing a successful authorization. Therefore, ADP instructs DID-Comm-API to respond to the SSI-Client with this successful authorization message.

21. DID-Comm-API responds to the SSI-Client as instructed.

22. The SSI-Client receives the response of the ACS, unpacks the DIDComm Message and parses the successful authorization message.

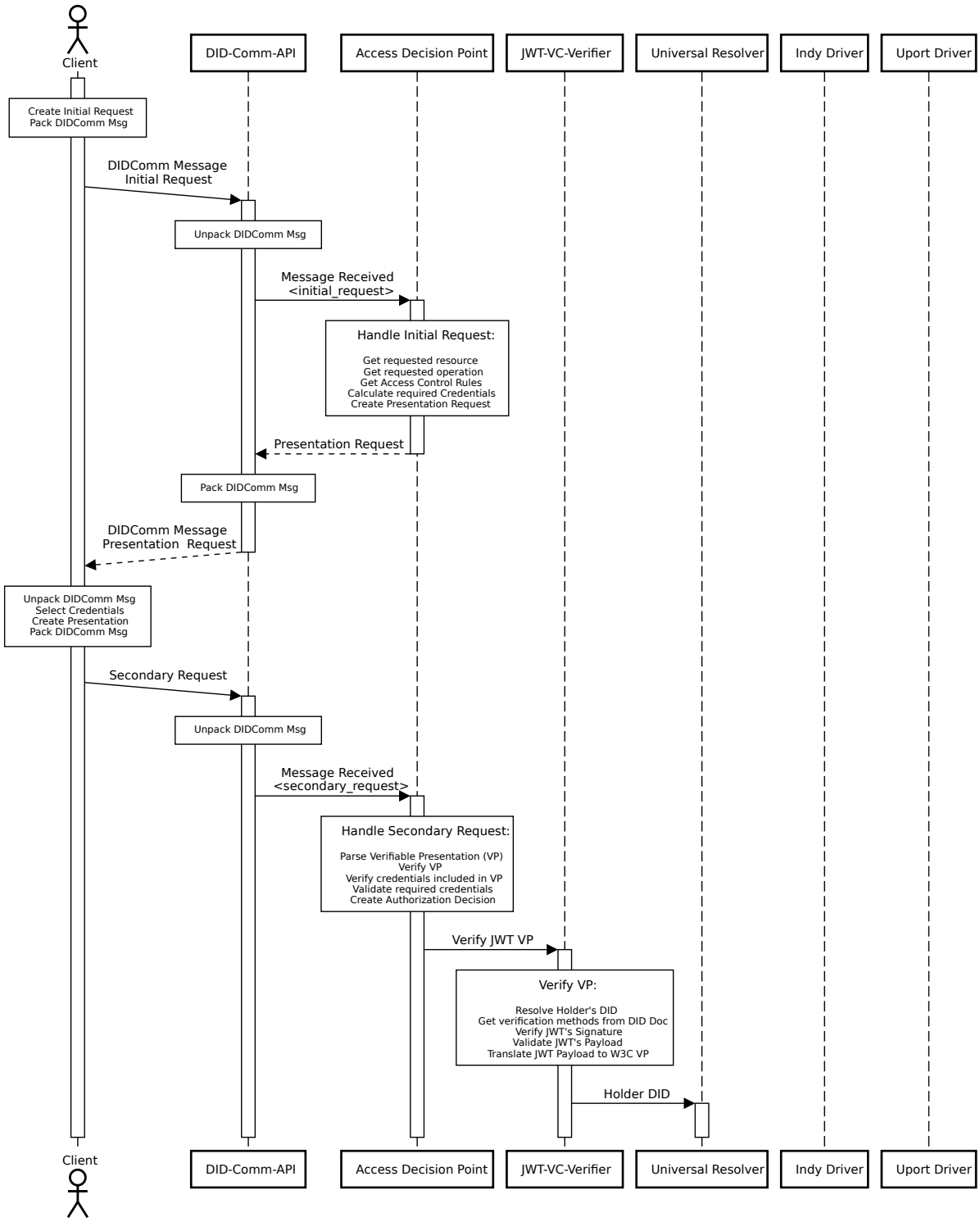


Figure 5.5: Authorization Process - Sequence Diagram (Part 1)

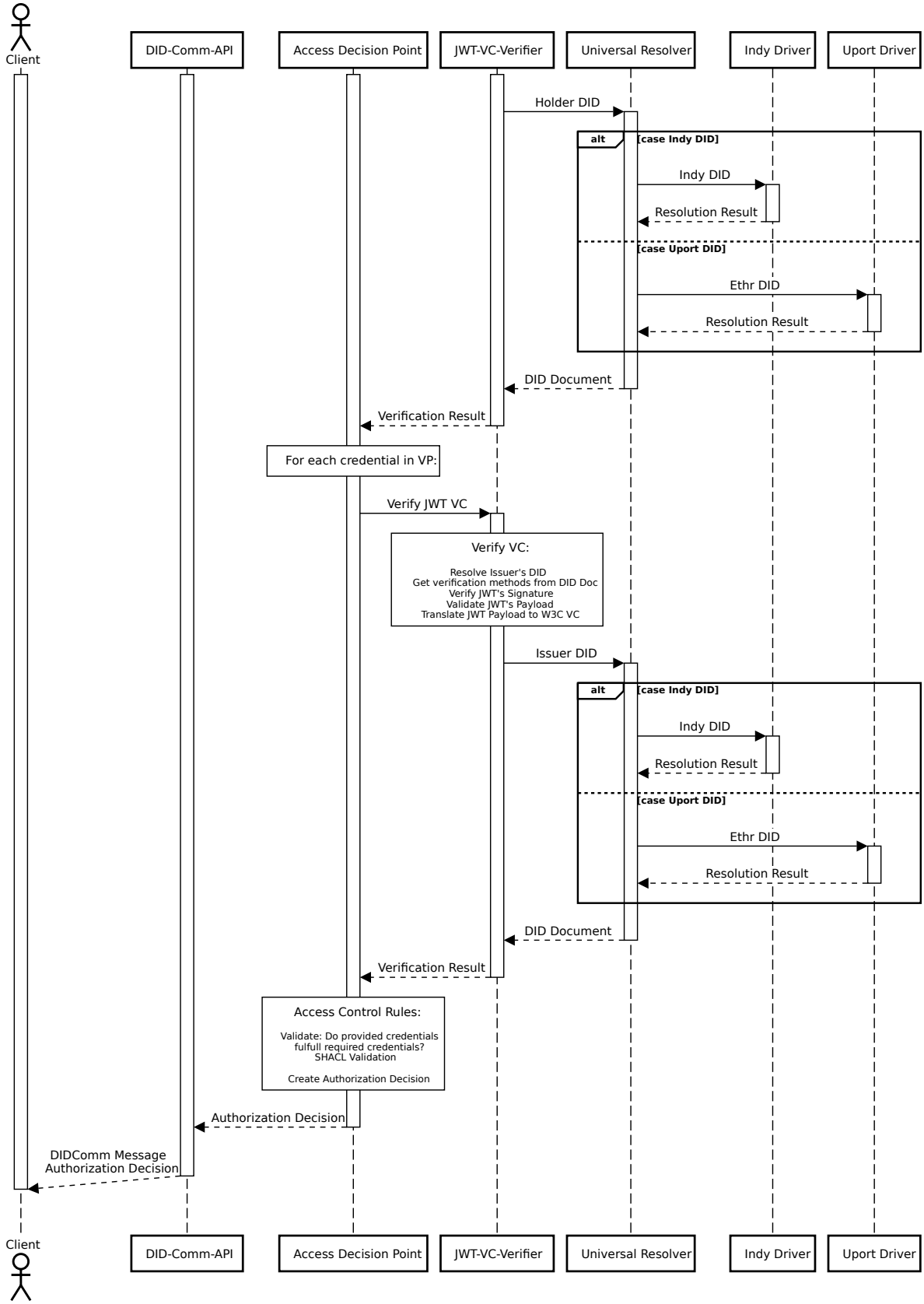


Figure 5.6: Authorization Process - Sequence Diagram (Part 2)

5.7 Implementation

As stated in the previous sections, a Proof-of-Concept prototype of the ACS and an example implementation of an SSI-Client have been implemented in order to prove the viability of the design and to evaluate the performance of the authorization process. We have contributed the implementations as public GitHub projects:

- SSI-based ACS - `https://github.com/vpapanchev/ssi-acs`. This project contains automated scripts for fetching, configuring and starting the entire microservice architecture of the ACS.
- DID Communication API - `https://github.com/vpapanchev/did-comm-api`.
- Verifier of JWT-based VCs - `https://github.com/vpapanchev/vc-jwt-verifier`.
- Access Decision Point - `https://github.com/vpapanchev/ssi-adp`.
- SSI-Client - `https://github.com/vpapanchev/ssi-acs-client`. The SSI-Client is implemented as a Python Flask API which is initialized with the URL and Peer-DID of the ACS and provides a single HTTP endpoint used to request a resource from the ACS. The project also contains automated scripts which can be used to send a single or multiple requests both sequentially or in parallel.

The reader is referred to the descriptions of the projects for more information about how to start and use the implementation. In following, some important open-source libraries are listed which have been used in our implementation:

- Python Peer-DIDs - `https://github.com/sicpa-dlab/peer-did-python` - a Python module providing support for the static layers (1, 2a and 2b) of the Peer-DID specification.
- Python DIDComm Messaging - `https://github.com/sicpa-dlab/didcomm-python` - a Python module implementing the DIDComm Messaging Protocol. The library provides support for packing and unpacking DIDComm Messages.
- RDFLib - `https://github.com/RDFLib/rdflib` - a Python package for working with RDF.
- PySHACL - `https://github.com/RDFLib/pySHACL` - a Python package providing validation of RDF graphs against Shapes Constraint Language (SHACL) graphs.
- PyJWT - `https://github.com/jpadilla/pyjwt` - a Python package for encoding and decoding JSON Web Tokens (JWTs)

- Libnacl - `https://github.com/saltstack/libnacl` - a Python wrapper of the nacl library via libsodium.
- Cryptography - `https://github.com/pyca/cryptography` - a Python cryptography package.

Evaluation

This chapter presents the evaluation of the designed SSI-based Access Control System (ACS). The evaluation results have been split into two categories - section 6.1 presents quantitative results measuring the performance of the authorization process, while section 6.2 summarizes a qualitative analysis of the ACS, e.g., about the achieved interoperability or the modelling of required credentials.

6.1 Quantitative Results

This section presents the quantitative results evaluating the performance of the system. The main objective of the evaluation is to measure the *End-to-End latency* of the ACS. The End-to-End latency (E2E-latency) is defined as the elapsed time starting when the user creates and sends the initial request for a given resource, and ending when the user receives the requested resource. In the implemented prototype, the resources are only an abstraction. Instead, the ACS only responds with a simple DIDComm Message notifying the SSI-Client of the successful authorization. In this way, the E2E-latency is composed of the following steps:

1. The SSI-Client creates a fresh Peer-DID used for the communication with the ACS.
2. The SSI-Client sends the initial request as a DIDComm Message to the ACS.
3. The ACS processes the initial request, creates a Verifiable Presentation Request (VPR) depending on the defined Access Control Rules and sends the VPR to the SSI-Client.
4. The SSI-Client parses the VPR and selects credentials from its wallet according to the defined SHACL description of the required credentials. Note that in a real scenario, the SSI-Client would/should prompt for user consent when sending the credentials. However, for a successful evaluation of the performance of the system, this user interaction is ignored.

5. The SSI-Client creates a Verifiable Presentation (VP) and sends it to the ACS.
6. The ACS parses the VP, verifies its cryptographic properties (which also includes resolving the Holder's DID and Issuers' DIDs) and validates whether the provided credentials satisfy the required ones.
7. The ACS responds to the SSI-Client with a DIDComm Message notifying the successful authorization.

Additionally to the E2E-latency, the throughput of the ACS needs to be evaluated. The throughput is measured as number of requests that the system can authorize per unit time in case that multiple requests are handled concurrently.

Due to the complex nature of the authorization process, measuring only the E2E-latency and the throughput would be insufficient to deduct many concrete conclusions for the overall performance of the ACS and how it can be improved. Therefore, we first analyze most of the procedures that are part of the entire authorization process and, thus, identify multiple bottlenecks for the performance. The results of the evaluation of these sub-procedures are presented in section 6.1.1. Optimization options for the bottlenecks, some of which have been implemented prior to measuring the E2E-latency and throughput, are presented and discussed in section 6.1.2. In the last two sections of the quantitative evaluation - 6.1.3 and 6.1.4, the measured E2E-latency and throughput are presented and compared to the theoretical expectations.

The evaluation was executed on a Virtual Machine (VM) running SMP Debian 4.9.320-2 (2022-06-30) x86_64. The VM has 4 CPU Cores (2.0Ghz) and 8GB RAM. As explained in the previous chapter, the ACS consists of multiple services, each implemented as a Docker container. The VM is running Docker version 19.03.14. Each evaluation result has been calculated as an average of multiple executions (exact number of executions is shown for each procedure).

6.1.1 Performance of Sub-Procedures

In this section, the performance of separate sub-procedures of the authorization process is evaluated.

Creating Peer-DIDs

In section 4.4.4, we have motivated using fresh Peer-DIDs for securing each separate outgoing communication. The SSI-Client uses this paradigm and creates a new Peer-DID every time a new resource is requested. The average time for generating a Peer-DID using the Python Peer-DID library is **2.281ms** which we evaluate as an **insignificant overhead**. The average time was calculated as the mean of 1000 executions.

Packing and Unpacking DIDComm Messages

All communication between the SSI-Client and the ACS is secured using the DIDComm Messaging Protocol. Figure 6.1 shows the linear growth (note the exponentially scaled X axis) of the average times for packing and unpacking DIDComm Messages relative to their size. For this evaluation, we have used DIDComm Messages containing a Verifiable Presentation (VP) and we have measured their sizes relative to the number of credentials (JSON-LD credentials expressed as JWTs) included in their VPs. The procedures of packing or unpacking a DIDComm Message containing between 1-128 JWT credentials using the Python DIDComm Messaging library require between **2-18ms**, which we evaluate as an **insignificant overhead**. The showed results were calculated as the average times of 1100 executions for each message size from which the fastest and slowest 50 times were discarded to avoid outliers.

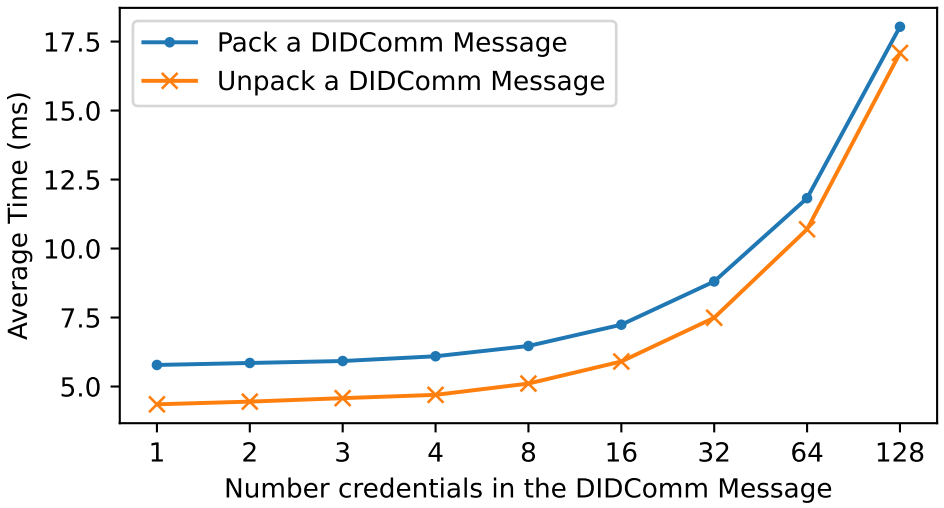


Figure 6.1: Pack and Unpack DIDComm Messages

Creating a Verifiable Presentation (using JSON-LD + JWT Credentials)

Once, the SSI-Client has selected which credentials to provide, the SSI-Client needs to create a Verifiable Presentation (VP) according to the VC Data Model specification 2.4.2, which includes digitally signing the presentation with the Holder's private key, and to pack the VP in a message conforming to the Present-Proof protocol 4.5.2. Figure 6.2 shows the performance of creating VPs relative to their size, once again measured as number of credentials included in the VP. Creating a VP containing 1-128 credentials requires **1-6ms** which we evaluate as an **insignificant overhead**. The showed results were calculated as the averages of 1000 executions for each number of credentials.

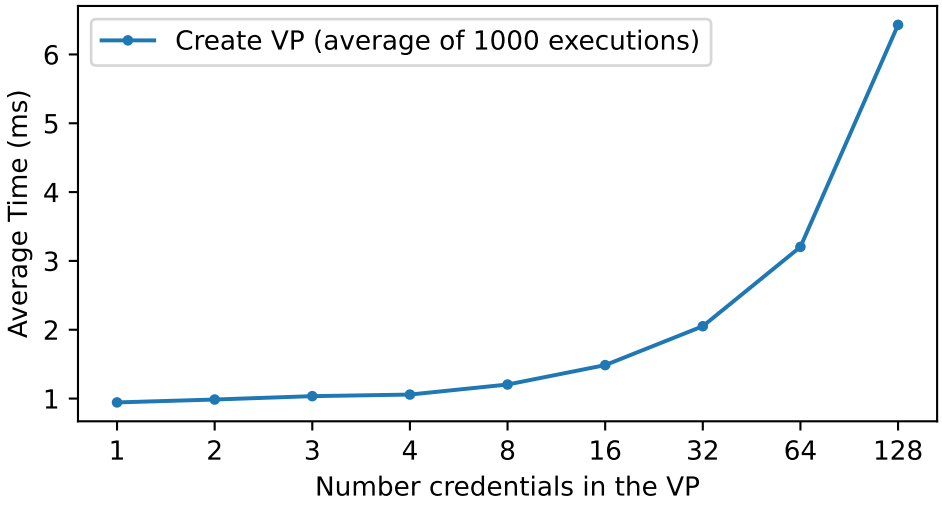


Figure 6.2: Create a Verifiable Presentation

Validate Credentials against SHACL Descriptions of Required Credentials

Given a LD-Credential and a SHACL description of a required credential, validate whether the credential satisfies the SHACL definition. This is one of the core procedures of the authorization process, as both the ACS and the SSI-Client need to execute it. The SSI-Client selects which credentials to provide by checking which of their credentials satisfy the presentation request of the ACS. On the other side, the ACS validates whether the credentials provided by the user satisfy the required ones. Table 6.1 shows the average times of this validation for the four different credential types we have defined. The rightmost cell of the table corresponds to the case where all 4 credentials are validated against a SHACL description requiring all of them. The validation of a single credential takes **between 1 and 1.5 seconds** which we evaluate as a **significant overhead**. The next results will show that the majority of the overhead is due to the parsing of the Linked Data Credentials as RDF Graphs which is a required step prior to the actual SHACL validation.

Credential Type	University Degree	Resident Card	Driving Licence	Covid-19 Vaccination	All 4 Together
Time (s)	1.449	1.094	1.103	1.139	5.378

Table 6.1: Validating SHACL Required Credentials

Parsing a LD-Credential as an RDF Graph

To execute the SHACL validation, both the LD-Credentials and the SHACL definitions need to be parsed as RDF Graphs which requires sending HTTP Requests to fetch the Linked Data contexts used by them. This operation depends heavily on where the contexts

are stored and how fast they are accessed. In our evaluation we have used 4 credential types - an example University Degree credential with a context hosted on the `https://www.w3.org/` domain, and a Resident Card, a Driving Licence and a Covid-19 Vaccination credentials with contexts hosted on the `https://w3id.org/` domain. Table 6.2 shows that the average time for parsing the University Degree credential is **1.385 seconds**, while the other credential types require **around 1 second**. Each of the results was calculated as the average between 110 executions from which the slowest and fastest 5 were discarded to avoid outliers. Due to the **significance of this overhead**, we determine that the procedure of parsing linked data credentials as RDF Graphs is one of the bottlenecks of the authorization process. Options for optimization of the procedure are discussed below.

Credential Type	University Degree	Resident Card	Driving Licence	Covid-19 Vaccination
Time (s)	1.385	1.003	1.007	1.009

Table 6.2: Parse a LD-Credential as an RDF Graph

Resolving a DID using the DIF Universal Resolver

To cryptographically verify a Verifiable Presentation (VP), the ACS needs to resolve the Holder's DID and the DIDs of the issuers of all credentials included in the VP. The ACS uses a local instance of the DIF Universal Resolver configured with two drivers - the Uport driver for Ethr-DIDs (we have used Ethr-DIDs rooted on the Ropsten Ethereum network) and the Indy-DID driver configured to connect to a local instance of the VON network. Table 6.3 shows that the average resolution time for **Indy-DIDs is 0.516 seconds** and for **Ethr-DIDs: 1.748 seconds**. As expected, the resolution times vary heavily for the different DID Methods. Even though, resolving Indy-DIDs is approximately 1.2 seconds faster than Ethr-DIDs, we consider both as a **significant overhead**. Options for optimization of this procedure are discussed below.

DID Method	Indy-DIDs	Ethr-DIDs
Time (s)	0.516	1.748

Table 6.3: Resolving DIDs with the Universal Resolver

6.1.2 Possible Optimizations

With the presented evaluation results so far, two bottlenecks of the authorization process have been identified - parsing Linked Data Credentials as RDF Graphs and resolving DIDs.

Firstly, it is important to note that the SSI-Client application has access to the locally stored credentials prior to any authorization. Therefore, the bottleneck of parsing the credentials as RDF Graphs on the client-side can be avoided by **parsing the credentials once during initialization of the SSI-Client**. In this way, when the SSI-Client needs to select credentials according to a VPR, the SSI-Client can directly use the already parsed and stored in memory RDF Graphs for the SHACL validation. This optimization option is critical as it might save up to $O(n)$ seconds per authorization process (where n corresponds to the number of credentials stored in the SSI-Client wallet). For the evaluation of the E2E-latency, we have implemented an SSI-Client which enforces this optimization.

The same optimization strategy cannot be transferred to the ACS-side directly as the ACS does not have access to the clients' credentials prior the actual authorization. However, multiple other optimization options can be used. First of all, the main overhead of parsing the credentials is the fetching of their contexts which is a network operation and does not require much CPU resources. Therefore, the ACS can be optimized to always **parse multiple credentials in parallel**. Furthermore, in many cases, the Access Control Rules specify concrete types of credentials including their contexts. Then, the credentials contexts used in any successful authorizations can be known prior to the actual authorizations. Therefore, the ACS can be further optimized to **cache all known and expected RDF contexts**, and to only fetch new contexts when required. These optimization strategies have not been implemented by our prototype. Evaluating the achieved performance with such optimizations can be regarded as future work.

The procedure of resolving DIDs depends on the DID Methods and the efficiency of their resolution drivers contributed to the Universal Resolver. It is worth mentioning, that the Universal Resolver is also still in development stages and some design decisions still need to be refined. One such design decision is the policy of the internal behaviors of the drivers as discussed in the GitHub issues of the project[1]. Depending on the driver implementation, a resolver can be "more or less close to the truth", depending on how the driver resolves the DIDs - ranging from having a direct access to a full node of the DID Method's DLT (very secure but heavyweight) up to querying a remote transaction explorer API (least secure but lightweight). Such design decisions also have direct impact on the performance of the Universal Resolver. It is, therefore, hard to conclude how the performance of the Universal Resolver will improve in the future. Currently, the only possible optimization option regarding the ACS is to provide more resources to the deployed Universal Resolver instance and to **execute any resolutions in parallel**, as they are theoretically completely independent procedures. Currently, our prototype

[1]Internal behavior of DID Drivers for the Universal Resolver - `https://github.com/decentralized-identity/universal-resolver/issues/4`

supports only sequential resolution of DIDs belonging to the same authorization process, e.g., sequentially resolving the DID of the Holder and then the DIDs of the issuers, and a parallel resolution of up to 2 DIDs belonging to separate authorization processes, i.e., when the ACS is handling multiple user requests in parallel.

6.1.3 End-to-End Latency

To measure the E2E-latency of the ACS, an SSI-Client project was implemented as a Flask API which can be initialized once and used many times afterwards to send user requests to the ACS. Sending requests in parallel using the SSI-Client is also possible. During the initialization of the SSI-Client, all credentials are parsed as RDF Graphs and stored in memory, as explained in the previous section. The SSI-Client is initialized with two separate wallets, both containing the same types of credentials. However, all DIDs used in the credentials of the first wallet are Indy-DIDs and all DIDs used in the credentials of the second wallet are Ethr-DIDs. Therefore, two types of requests are defined:

- **Indy-based Requests:** The Holder DID and the DIDs of all Issuers are Indy-DIDs.
- **Ethr-based Requests:** The Holder DID and the DIDs of all Issuers are Ethr-DIDs.

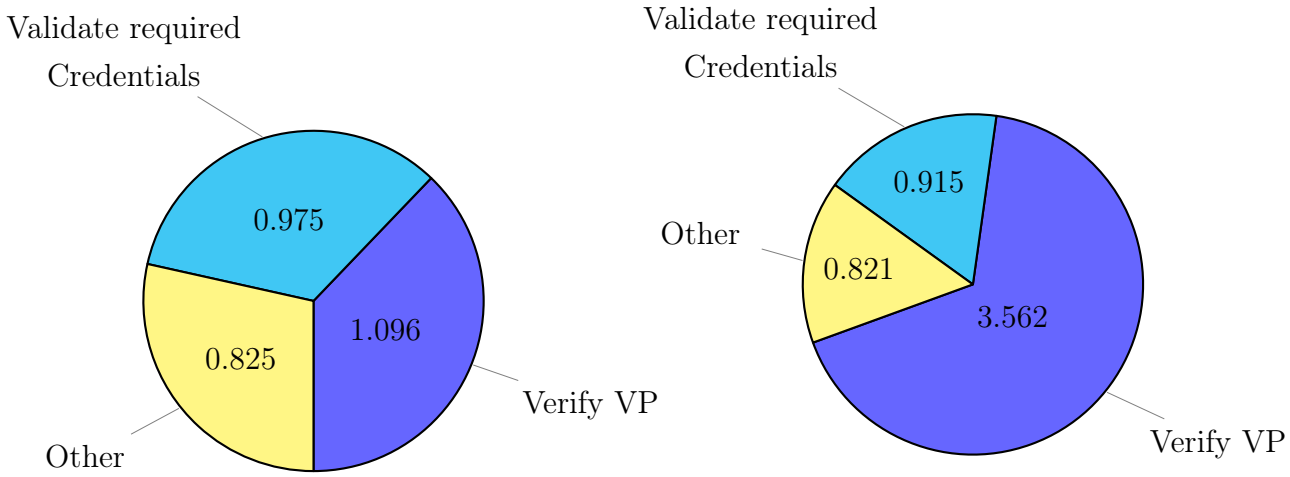
In this way, we can measure the E2E-latency separately for the two DID Methods.

Furthermore, each user request requests such a resource, that requires exactly a single credential to be provided. Therefore, every request is authorized by providing a single VP containing a single VC and the verification of this presentation requires exactly 2 DID resolutions - one for the holder and another one for the issuer of the credential. Each of the two wallets is initialized with a single University Degree credential, two Resident Card credentials, one Driving Licence credential and one Covid-19 Vaccination credential - in total 5 Linked Data credentials expressed as JWTs (JSON-LD + JWT, see section 4.3). We define the term **1 batch of requests** as sending the 10 possible different requests - 5 Indy-based requests (1 for each credential) and 5 Ethr-based requests (1 for each credential).

To evaluate the E2E-latency we used an automated script which sends (completely sequentially) 100 batches of requests - in total 500 Indy-based requests and 500 Ethr-based requests. Table 6.4 shows that the average **E2E-latency for Indy-based requests is 2.896 seconds** while the average **E2E-latency for Ethr-based requests is 5.298 seconds** - around 2.4 seconds higher. Figure 6.3 shows two pie charts comparing the overhead of the verification of the VP (which includes resolving two DIDs) and the validation of the required credentials for Indy- and Ethr-based requests. It is clear that the 1.2 seconds slower resolution of Ethr-DIDs is responsible for the higher E2E-latency of the Ethr-based requests.

Request Types	Indy-based Requests	Ethr-based Requests	Both Request Types
Time (s)	2.896	5.298	4.097

Table 6.4: End-to-End Latency



(a) Indy-based Requests E2E-latency

(b) Ethr-based Requests E2E-latency

Figure 6.3: Composition of the E2E-latency

It is worth repeating, that when measuring this E2E-latency, the 2 DID resolutions of a single request are executed sequentially and the ACS fetches the Linked Data contexts during handling the request, i.e., the expected Linked Data contexts have not been cached. Resolving the DIDs in parallel should save up to 0.516 seconds for Indy-based requests and up to 1.748 seconds for Ethr-based requests (see table 6.3). Additionally, up to 1 second can be saved if the ACS caches the expected LD contexts (see table 6.2). Under these optimizations, the E2E-latency of the authorization process can be improved to **1.4 seconds for Indy-based requests** and **2.6 seconds for Ethr-based requests**.

6.1.4 Throughput

We have also evaluated the performance of the ACS under parallel requests. Each of the deployed Docker containers was initialized with 2 worker threads, therefore, the handling of multiple requests can make progress concurrently. Furthermore, during blocking operations, such as fetching Linked Data namespaces, other requests are handled in parallel. As already mentioned, our locally-deployed instance of the Universal Resolver can handle two DID resolution requests in parallel. Because the procedure of a DID resolution has the slowest average time of

$$1/2 * mean(1.096, 3.562) \approx 1.15$$

seconds (mean of the average resolution times for Indy-DIDs and Ethr-DIDs), and every request requires resolution of 2 DIDs, we expected that the theoretical throughput should be approximately 1.15 seconds - the average time for a single DID resolution.

Table 6.5 shows the measured throughput of the system under 10, 20 and 50 requests in parallel. Each result was calculated as the average of 15 repetitions. As analyzed theoretically, the **throughout per request approximates 1.15 seconds**.

Number Requests in Parallel	10	20	50
Total Time (s)	12.228	23.198	57,512
Time per 1 request (s)	1.223	1.160	1.150

Table 6.5: Throughput of the ACS

6.2 Qualitative Results

This section provides a qualitative evaluation in the form of a summarized analysis of some aspects of the designed SSI-based Access Control System, such as the achieved interoperability properties and a comparison of the alternatives for the modeling of required credentials.

6.2.1 Interoperability of DID Methods

In the scope of the ACS, it is sufficient to be able to resolve DIDs of different DID Methods in order to achieve the defined interoperability goals. This is because the ACS only requires read operations on the cryptographic materials stored in the DID Documents in order to successfully verify a credential or presentation submitted by the SSI-Client.

The usage of a locally deployed instance of the DIF Universal Resolver provides the functionality of resolving DIDs of multiple DID Methods. Although, not all DID Methods provide an open-source driver implementation that can be deployed and used by the Universal Resolver, a sufficient percentage of the most commonly used DID Methods do. Furthermore, we expect that with the further development of the Universal Resolver and possibly its integration in more deployed applications, other DID Methods will also be incentivized to provide drivers.

Deploying a local instance of the Universal Resolver has the additional benefit that the deploying system has full freedom to configure the supported DID Methods and to scale

the performance of the deployed instance depending on the system‘s load. We expect that in a realistic scenario, systems will initially support only a small number of DID Methods which can later be extended (for example depending on user demand) by deploying and configuring additional drivers for other DID Methods.

As mentioned in the previous sections, currently our implementation of DID-Comm-API only supports Peer-DIDs for the communication using the DIDComm Messaging protocol. However, it should be up to the End-Users to decide whether to use newly generated Peer-DIDs or their long-lived DIDs for the communication (assuming these DIDs have been updated accordingly to contain a DIDComm Messaging Service Endpoint as explained in section 4.4.4). Although not implemented, simply connecting DID-Comm-API to the Universal Resolver and, thus, implementing a local resolver interface to be supplied to the DIDComm Messaging protocol implementation used inside the DID-Comm-API component, will be sufficient to also support different types of DID Methods for the communication.

As a summary, we evaluate that the Universal Resolver can be used to achieve a solution to the SSI-Interoperability problem of supporting multiple DID Methods.

6.2.2 Interoperability between VCs and different DID Methods

During our initial research, it was unclear to us whether the practical interoperability between VCs and different types of DIDs is possible, i.e., whether an Issuer using DIDs of one DID Method can issue a credential to a Holder having a DID of another DID Method.

We have successfully created JSON-LD +JWT VCs issued by an Issuer using an Ethr-DID 4.2.2 anchored on the Ropsten test network to a Holder using an Indy-DID 4.2.3 on a locally deployed VON network (and vice versa). With this, we prove that the VC Data Model [105] and the DID [104] specifications allow for such interoperability.

However, we have also tried to do the same with KILT DIDs 4.2.4. For creating and managing the KILT DIDs we used the provided open-source KILT SDK. Because the SDK does not provide a direct access to the private keys controlling the KILT DIDs and getting such access requires a high-overhead update of the SDK (for more information see section 4.2.4), we have been unsuccessful in using KILT DIDs with any type of credentials other than the KILT-specific credential types.

It is also worth mentioning that when deploying a local Indy ledger using the VON Network[2] and then publishing Indy-DIDs on this ledger using the provided command-line interface, access to the private keys controlling the created DIDs is also not directly given. However, with a simple update of the function used when generating new DIDs, the

[2] `https://github.com/bcgov/von-network` - VON Network Github

private keys can be accessed. Furthermore, the user can alternatively publish DIDs only using a cryptographic seed which is calculated from the public key. With the knowledge of the used key generation algorithm, the user can generate a public and private key pair prior to the actual publishing of the DIDs and then register a new DID using this key pair. Thus, the registered DID is controlled by the private key which is already known to the user.

As a summary, although the underlying specifications are well-suited for this type of interoperability, it is up to the organizations developing the specific DID Methods and their products to guarantee the practical feasibility of such interoperability approaches. Furthermore, we have only practically tested this interoperability using JWT-based credentials. Some of the other credential flavors presented in section 4.3, e.g., JSON-ZKP-CL Anonymous Credentials, should be further evaluated, as they are significantly more coupled to an underlying Verifiable Registry.

6.2.3 Interoperability of different VC Formats

According to our view about an SSI-Interoperable ACS, if an End-User presents a valid VC that satisfies the defined authorization rules, the End-User should be granted access to the requested resource independent of the type of their credential (assuming the authorization rules do not explicitly require a specific credential type). For this to be possible, the ACS should support the procedures of requesting, exchanging and verifying multiple (or even all) formats of VCs.

In section 4.3, the differences between the currently existing formats of VCs were analyzed. The theoretical system design is also intended for such interoperability by integrating multiple (independent of each other) components, each responsible for handling a specific type of credentials. Due to the standardized data format of VCs and the usage of semantic technologies (including our semantic Access Control Rules), we evaluate that such interoperability between different VC flavors (using Linked Data payloads) is indeed possible, as long as verifiers for the different types of credentials are implemented and deployed. However, due to time constraints, we have implemented only a single verifier component supporting the JSON-LD VCs using JSON Web Tokens as cryptographic proofs (JSON-LD + JWT). Therefore, building a Proof-of-Concept realization of an ACS which is interoperable with multiple credential formats, and evaluating its properties, e.g., comparing performance between the different credential formats, has not been done in the scope of this work and remains as a possible future work.

6.2.4 Modelling Required Credentials in the Authorization Rules

An important part of every access control system is the way the access logic is defined. In our case, we wanted to extend the Web Access Control (WAC) specification 3.2.2 used by the Solid protocol with the additional functionality of modelling VCs that the End-Users are required to present. For the modelling of the credentials the following requirements were defined:

- **Functionality** - describing the type of the credential, the information included in the credential, and combinations of credentials using the logical AND and OR operations, e.g., "*provide credential A **AND** (credential B **OR** credential C)*"
- **Semantic Payloads** - in the case of Linked Data credentials, the modelling of the required credentials should be capable of understanding and using the RDF standard in order to specify semantic requirements on the credential payloads.
- **Human-understandable Design** - in many cases the authorization rules are written and managed directly by the Resource Owners. For example, in Solid each user can have their own Solid POD and can define the access control policies for their resources by writing the authorization rules by themselves. Furthermore, End-Users wanting to access a given resource are often given the authorization rules for this resource in order to "understand" how they should get access to it. Therefore, having an easy-to-read design for the modelling of the credentials is an important requirement.

In the scope of the work, the following approaches for modelling the required credentials as an extension to the WAC authorization rules are evaluated and compared:

- **DIF Presentation Exchange** - directly modelling the required credentials using Presentation Definition objects as used when exchanging credentials using the DIF PE protocol (see section 4.5.1).
- **SPARQL** - executing SPARQL queries (see section 2.5.2) on the Linked Data payloads of the credentials and describing requirements on the results returned by those queries.
- **ShEx** - describing the Linked Data payloads of the credentials using the ShEx language (see section 2.5.3).
- **SHACL** - describing the Linked Data payloads of the credentials using the SHACL language (see section 2.5.4).

As explained in section 4.5.1, the Presentation Definition objects used in the DIF Presentation Exchange protocol are fairly complex and their JSON structure grows very fast which makes them hard to read and understand. Furthermore, they do not leverage any RDF semantics when referencing and describing values stored in the credentials. Therefore, we have decided against this option.

Directly using SPARQL 2.5.2 to query the payload of the credentials and to then evaluate the returned results by the queries, e.g., by comparing them with specific expected values, was our initial idea on how to model the required credentials. This approach does exploit the RDF standard and, in our opinion, SPARQL queries are easy to read and understand for users with some experience with semantic technologies. The drawback of this approach is that additionally to the SPARQL queries, the evaluation of the returned results also needs to be modelled, i.e., only using SPARQL queries is not sufficient.

Both ShEx 2.5.3 and SHACL 2.5.4 are languages for describing RDF Graphs and validating them against the defined set of conditions. We have used both of them to successfully model a VC with a Linked Data payload. Furthermore, achieving the logical AND and OR operations can be done by supplying multiple ShEx or SHACL descriptions in the same WAC authorization rule (AND) or in multiple authorization rules (OR). Furthermore, both ShEx and SHACL are easier to write and read compared to the Presentation Definitions of the DIF PE protocol. An example modeling of the same *University Degree* credential certifying a *Bachelor Degree of Science and Arts* using SHACL and ShEx is given in listings 6.1 and 6.2 respectively. In the end, we decided to use SHACL for the modelling of required credentials due to the following reasons:

- SHACL is an official W3C Recommendation, while ShEx is only a Community Group Effort. Therefore, it is expected that SHACL will be supported by more systems.
- SHACL also allows for integrating SPARQL queries as part of the SHACL descriptions. Therefore, with SHACL the Resource Owner decides by themselves whether to use the built-in primitives of SHACL or SPARQL queries to reference specific parts of the RDF Graphs representing the Linked Data credentials.
- The validation results of SHACL have a standardized RDF-based structure showing why the RDF graph failed validation. This can be used to provide feedback to the End-Users in case their credentials do not satisfy the defined requirements. In comparison, the validation results of ShEx are not standardized and provide less information on the reasons for the unsuccessful validation.
- SHACL has a single syntax - RDF in any serialization, which makes it trivial to be integrated as an extension to the WAC authorization rules. The syntax used in the

ShEx example is the compact ShEx syntax which is not RDF-based. ShEx has an RDF-based syntax but is fairly more complex and hard to read and understand.

```
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix schema: <http://schema.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix cred: <https://www.w3.org/2018/credentials#> .
@prefix ex: <https://example.org/examples#> .

aifb:BachelorDegreeCredentialShape
    a sh:NodeShape ;

    # Objects of type cred:VC are validated against this NodeShape
    sh:targetClass cred:VerifiableCredential ;

    # Required class of the credential
    sh:class ex:UniversityDegreeCredential ;

    # Required class of the credentialSubject -> degree property
    sh:property [
            sh:path (cred:credentialSubject ex:degree) ;
            sh:class ex:BachelorDegree ;
    ] ;

    # Required value of credentialSubject->degree->name
    sh:property [
            sh:path (cred:credentialSubject ex:degree schema:name) ;
            sh:hasValue "Bachelor of Science and Arts"^^rdf:HTML ;
    ] .
```

Listing 6.1: SHACL Description of a University Degree Credential

```
PREFIX schema: <http://schema.org/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX cred: <https://www.w3.org/2018/credentials#>
PREFIX ex: <https://example.org/examples#>

start = @:CredentialShape

```

```
:CredentialShape {
    # the type of the credential
    a [cred:VerifiableCredential] ;
    a [ex:UniversityDegreeCredential] ;
    # Requirements on the credentialSubject -> degree property
    cred:credentialSubject {
        ex:degree {
                a [ex:BachelorDegree] ;
                schema:name ["Bachelor of Science and Arts"^^rdf:HTML]
        }
    }
}
```

Listing 6.2: ShEx Description of a University Degree Credential

Summary and Outlook

This chapter summarizes the results from the previous chapters and presents some topics that can be further analyzed in future research.

7.1 Summary

In this thesis, the current state of Self-Sovereign Identity (SSI) was analyzed and an interoperable SSI-based Access Control System (ACS) was designed using a Microservice architecture which allows for flexible substitution and extensibility of individual parts of the system. Furthermore, multiple services are internally-represented as driver-based architectures which makes possible an incremental extensibility of the supported SSI technologies. An open-source Proof-of-Concept prototype (PoC-Prototype) of the system was implemented and its performance was evaluated. The communication between the ACS and the users was secured using the DIDComm Messaging Protocol in combination with Peer-DIDs. The ACS achieves interoperability with multiple DID Methods by deploying a local instance of the DIF Universal Resolver. Interoperability of multiple types of Verifiable Credentials is also regarded in the ACS design by employing a driver-based architecture, however, the implemented PoC-Prototype currently only supports JSON-LD Credentials expressed as JSON Web Tokens. The Access Control Logic of the system is based on the semantic Web Access Control (WAC) specification used by the Solid protocol. Multiple approaches for modelling required credentials as part of the WAC authorization rules were analyzed, including using the SHACL language to model the RDF graphs represented by the Linked Data credentials. The End-to-End latency of the authorization process was evaluated and requires between 2-5 seconds depending on the used DID Methods but can (theoretically) be further optimized to 1.5-3 seconds.

7.2 Future Work

As noted in the previous chapters, for the purposes of rapid prototype development, our Proof-of-Concept implementation does not fully correspond to the defined design. Therefore, some components need to be updated and new ones implemented. Additionally to the implementation, further research can be done in the direction of how to integrate an SSI-based Access Control System in existing implementations, such as resource servers with an already functioning (non-SSI-based) access control. The purpose of this section is to present possible directions for future work in the realm of an interoperable Access Control System based on Self-Sovereign Identities.

7.2.1 Future Work w.r.t. Interoperability

The implemented Proof-of-Concept prototype currently only supports JSON-LD Credentials expressed as JSON Web Tokens (JSON-LD + JWT). We regard the implementation of additional drivers for verification of other types of Verifiable Credentials as one of the most important future work directions due to the following reasons:

- **Practical Interoperability of VCs** - by implementing additional drivers for other VC types, the system can capitalize on the driver-based architecture in practice.
- **Comparison between different VCs** - various aspects of the different types of VCs can be evaluated. Examples include the performance of verification, complexity of use and different levels of achieved privacy. Furthermore, some VC types, e.g., the Anonymous Credentials used in HL Indy and Aries, are more coupled to an underlying Verifiable Registry and might prove more difficult for integration.
- **Easier Adoption of SSI Technologies** - providing open-source implementations of additional drivers should simplify deployment of SSI technologies for the service providers.

Additionally to the interoperability of Verifiable Credentials, drivers for resolution of more DID Methods can be deployed and connected to the Universal Resolver in order to further evaluate the differences between their performance. Also, it will be beneficial to connect the DID Communication API with the Universal Resolver and provide a DID Resolution interface that connects to the implemented DIDComm Messaging component. In this way, multiple DID Methods (and not only Peer-DIDs) can be supported for securing the communication between the Access Control System and the SSI-Clients. This will improve the interoperability of the system with users that want to use their long-term identifiers for securing the communication.

Furthermore, depending on how the service provider publishes the Server-DID (currently a Peer-DID generated by the DID Communication API) used by the SSI-Clients to communicate with the Access Control System, it might be beneficial for some use-cases to use a long-lived and well-known DID of the service provider. This, however, requires that the SSI-Clients are also capable of resolving the DID Method used by the service provider. Further research of this topic is another possible future work direction.

7.2.2 Optimizations of the Authorization Process

As explained in section 6.1.2, the performance of the authorization process can be further improved by employing massive parallel resolution of DIDs (including DIDs belonging to the same user request) and server-side caching of expected Linked Data contexts. These optimization strategies can be implemented as a future work in order to analyze their effect on the End-to-End latency and Throughput of the authorization process. Additionally, further research of possible optimizations and their employment can also yield important results.

7.2.3 Support for DIF Presentation Exchange

The design of the Access Control System (see section 5.4) states that the DIF Presentation Exchange 4.5.1 format of Verifiable Presentation Requests (VPRs) should also be supported, as this would improve the interoperability of the system with existing SSI Agent implementations due to the wide adoption of the protocol. However, the implemented Proof-of-Concept prototype currently only uses VPRs based on our modelling of required credentials using SHACL. The Access Control System uses the HL Aries Present Proof Protocol 4.5.2 which allows for multiple formats of VPRs to be used independently of each other. However, research in the direction of implementing an automated mapping or translation of the SHACL required credentials to Presentation Definition objects used by the DIF Presentation Exchange protocol remains an open problem that can be further examined in future work.

7.2.4 Man-in-the-Middle Attacks

As presented in section 4.4.5, prevention of MITM attacks on the DIDComm Messaging protocol is a complex topic which requires application-specific handling methods and further research. The most popular prevention mechanism is to use VCs in order to prove that the currently used DIDs for the communication belong to the appropriate identities. This approach requires requesting and verifying VCs, as well as maintaining

a list of trusted issuers for these VCs, therefore, it is best suited to not be implemented on the DIDComm Messaging layer but on the layers above. Furthermore, research in the direction of protecting the VCs used to prove the absence of a MITM attack is also necessary.

7.2.5 Integration Strategies for the Access Control System

Another important future work direction is the research and evaluation of different strategies for the integration of the designed Access Control System in a realistic scenario with an already existing resource server. One possibility is to update the DID Communication API component to directly provide the requested resources following a successful authorization. Alternatively, the Resource Server can forward user requests to the Access Control System by providing some QR code that the users need to scan with their mobile SSI agents in order to perform the authentication and authorization. Examining such integration strategies for different use-cases is important especially if assumed that most service providers already have an existing infrastructure (such as a gateway to their Resource Server) and would strive for SSI integration requiring minimal changes.

Appendix

```
{
    "@type": "https://didcomm.org/present-proof
                /%VER/request-presentation",
    "@id": "<uuid-request>",
    "goal_code": "<goal-code> | Optional",
    "comment": "some comment | Optional",
    "will_confirm": false,
    "present_multiple": false,
    "formats" : [
        {
            "attach_id" : "<attach@id value>",
            "format" : "<format>",
        }
    ],
    "request_presentations~attach": [
        {
            "@id": "<attachment identifier>",
            "mime-type": "application/json",
            "data":  {
                "base64": "<base64 encoded VPR>"
            }
        }
    ]
}
```

Listing 1: Format of the Presentation-Request message used in HL Aries Present Proof Protocol

```
{
    "@type": "https://didcomm.org/present-proof
                /%VER/presentation",
    "@id": "<uuid-presentation>",
    "goal_code": "<goal-code> | Optional",
    "comment": "some comment | Optional",
    "last_presentation": true,
    "formats" : [
        {
            "attach_id" : "<attach@id value>",
            "format" : "<format>",
        }
    ],
    "presentations~attach": [
        {
            "@id": "<attachment identifier>",
            "mime-type": "application/json",
            "data": {
                "_comment": "The VPR is included here in the
                                specified format, e.g., as a JWT:"
                "jwt": "eyJh...bGsw5c",
            }
        }
    ]
}
```

Listing 2: Format of the Presentation message used in HL Aries Present Proof Protocol

```
{
    "sender": "Communication DID of Sender (Peer-DID)",
    "request": {
        "type": "DIDComm/HTTP",
        "http_request_method": "GET/PUT/POST/..",
        "message": {
            "id": "DIDComm Message ID",
            "type": "DIDComm Message Type",
            "body": "DIDComm Message Body",
            "attachments": [
                "DIDComm Message Attachment"
            ]
        }
    },
}
```

Listing 3: ADP Message Handler API: Format of the New User Request Received Webhook

```
{
    "response": {
        "type": "DIDComm/HTTP",
        "http_code": "HTTP Status Code of response 200/400/401/..",
        "message": {
            "id": "Response DIDComm Message ID",
            "type": "Response DIDComm Message Type",
            "body": "Response DIDComm Message Body"
        }
    }
}
```

Listing 4: ADP Message Handler API: Response of ADP for New User Request Received

```
{
    "valid": "True/False",
    "error": "Error in case of unsuccessful verification",
    "data": {
        "payload": "The decoded JWT payload",
        "issuer": "DID of the issuer of the VC",
        "jwt": "The orignal JWT",
        "verifiableCredential": "The VC in the W3C data format"
    }
}
```

Listing 5: JWT VC Verifier: Format of a VC-Verification Response

```
{
    "valid": "True/False",
    "error": "Error in case of unsuccessful verification",
    "data": {
        "payload": "The decoded JWT payload",
        "holder": "DID of the issuer of the JWT (Holder DID)",
        "jwt": "The original JWT",
        "verifiablePresentation": "The VP in the W3C data format",
        "challenge": {
            "nonce": "The nonce contained in the VP",
            "domain": "The Domain specified in the VP"
        }
     }
 }
```

Listing 6: JWT VC Verifier: Format of a VP-Verification Response

Bibliography

[1] Acquisti, A., and Grossklags, J. Privacy and rationality in individual decision making. *IEEE security & privacy 3*, 1 (2005), 26–33.

[2] Adleman, L. M., and McCurley, K. S. Open problems in number theoretic complexity, ii. In *International Algorithmic Number Theory Symposium* (1994), Springer, pp. 291–322.

[3] Allen, C. *The path to Self-Soverereign Identity.* http://www.lifewithalacrity.com/2016/04/the-path-to-self-soverereign-identity.html.

[4] Allen, C., Duffy, K. H., Grant, R., and Pape, D. Btcr did method. Draft community group report, W3C, Aug. 2019. `https://w3c-ccg.github.io/didm-btcr/`.

[5] Antonino, A., and Nalpas, M. Kilt decentralised identifiers (did) method specification. Github specification, KILT Protocol, Mar. 2022. `https://github.com/KILTprotocol/kilt-did-driver/blob/master/docs/did-spec/spec.md`.

[6] Axon, L., and Goldsmith, M. Pb-pki: A privacy-aware blockchain-based pki. In *SECRYPT* (2017), pp. 311–318.

[7] Baars, D. Towards self-sovereign identity using blockchain technology. Master's thesis, University of Twente, 2016.

[8] Baker, T., and Prud'hommeaux, E. Shape expressions (shex) primer. W3c draft community group report, W3C, May 2022. `https://shexspec.github.io/primer/`.

[9] Belchior, R., Putz, B., Pernul, G., Correia, M., Vasconcelos, A., and Guerreiro, S. Ssibac: self-sovereign identity based access control. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)* (2020), IEEE, pp. 1935–1943.

[10] Berners-Lee, T., Cailliau, R., Luotonen, A., Nielsen, H. F., and Secret, A. The world-wide web. *Communications of the ACM 37*, 8 (1994), 76–82.

[11] Berners-Lee, T., Fielding, R., and Masinter, L. Uniform resource identifiers (uri): Generic syntax. Rfc 2396, IETF, Aug. 1998. `https://www.ietf.org/rfc/rfc2396.txt`.

[12] Berners-Lee, T., Hendler, J., and Lassila, O. The semantic web. *Scientific american 284*, 5 (2001), 34–43.

[13] Bhattacharya, M. P., Zavarsky, P., and Butakov, S. Enhancing the security and privacy of self-sovereign identities on hyperledger indy blockchain. In *2020 International Symposium on Networks, Computers and Communications (ISNCC)* (2020), IEEE, pp. 1–7.

[14] Bizer, C., Heath, T., and Berners-Lee, T. Linked data: Principles and state of the art. In *World wide web conference* (2008), vol. 1, p. 40.

[15] Bizer, C., Heath, T., and Berners-Lee, T. Linked data: Principles and state of the art. In *World wide web conference* (2008), vol. 1, p. 40.

[16] Bizer, C., Heath, T., Idehen, K., and Berners-Lee, T. Linked data on the web (ldow2008). In *Proceedings of the 17th international conference on World Wide Web* (2008), pp. 1265–1266.

[17] Brands, S. A technical overview of digital credentials.

[18] Brough, A. R., and Martin, K. D. Consumer privacy during (and after) the covid-19 pandemic. *Journal of Public Policy & Marketing 40*, 1 (2021), 108–110.

[19] Buchner, D., Zundel, B., Riedel, M., and Duffy, K. H. Presentation exchange 2.0.0. Working group draft, DIF. `https://identity.foundation/presentation-exchange/`.

[20] Buterin, V., et al. A next-generation smart contract and decentralized application platform. *white paper 3*, 37 (2014), 2–1.

[21] Callas, J., Donnerhacke, L., Finney, H., Shaw, D., and Thayer, R. Openpgp message format. Rfc 4880, IETF, Nov. 2007. `https://www.ietf.org/rfc/rfc4880.txt` Section 11. Packet Composition.

[22] Camenisch, J., and Lysyanskaya, A. A signature scheme with efficient protocols. In *International Conference on Security in Communication Networks* (2002), Springer, pp. 268–289.

[23] Capadisli, S. Web access control. W3c editor's draft, W3C Solid Community Group, Mar. 2022. https://solid.github.io/web-access-control-spec/.

[24] Capadisli, S., Berners-Lee, T., Verborgh, R., and Kjernsmo, K. Solid protocol. Work in progress protocol, W3C Solid Community Group, Dec. 2021. https://solidproject.org/TR/protocol.

[25] Caronni, G. Walking the web of trust. In *Proceedings IEEE 9th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2000)* (2000), pp. 153–158.

[26] Cohen, G. *Medium Article: Presentation Exchange: A Leap Toward Interoperability for the Decentralized Identity Ecosystem.* https://medium.com/decentralized-identity/presentation-exchange-a-leap-toward-interoperability-for-the-decentralized-identity-ecosystem-3cca03ef53c5, 2020.

[27] Conner Fromknecht, Dragos Velicanu, S. Y. A decentralized public key infrastructure with identity retention. Cryptology ePrint Archive, Report 2014/803, 2014. https://ia.cr/2014/803.

[28] *Covid Credentials Initiative.* https://www.covidcreds.org/.

[29] Curran, S., Bastian, P., and Hardman, D. Indy did method. Tech. rep., Hyperledger. https://hyperledger.github.io/indy-did-method/.

[30] Curren, S., Looker, T., and Terbu, O. Didcomm messaging. Specification: Editor's draft, DIF: Decentralized Identity Foundation, 2021. https://identity.foundation/didcomm-messaging/spec/.

[31] Der, U., Jähnichen, S., and Sürmeli, J. Self-sovereign identity – opportunities and challenges for the digital revolution. *arXiv preprint arXiv:1712.01767* (2017).

[32] Deventer, O., Lundkvist, C., Csernai, M., Hartog, K. D., Sabadello, M., Curren, S., Gisolfi, D., Varley, M., Hammann, S., Jordan, J., Harchandani, L., Fisher, D., Looker, T., Zundel, B., Curran, S., Rybas, D., and Shcherbakov, A. Peer did method specification. W3c document, DIF: Decentralized Identity Foundation, Oct. 2021. https://identity.foundation/peer-did-method-spec/.

[33] Dhamija, R., and Dusseault, L. The seven flaws of identity management: Usability and security challenges. *IEEE Security & Privacy 6*, 2 (2008), 24–29.

[34] *DIF Universal Resolver.* https://github.com/decentralized-identity/universal-resolver.

[35] Diffie, W., and Hellman, M. New directions in cryptography. *IEEE Transactions on Information Theory 22*, 6 (November 1976), 644–654.

[36] Diffie, W., and Hellman, M. E. New directions in cryptography. In *Secure communications and asymmetric cryptosystems*. Routledge, 2019, pp. 143–180.

[37] Duggal, R., Brindle, I., and Bagenal, J. Digital healthcare: regulating the revolution, 2018.

[38] Durnell, E., Okabe-Miyamoto, K., Howell, R. T., and Zizi, M. Online privacy breaches, offline consequences: Construction and validation of the concerns with the protection of informational privacy scale. *International Journal of Human–Computer Interaction 36*, 19 (2020), 1834–1848.

[39] Dwork, C., and Naor, M. Pricing via processing or combatting junk mail. In *Annual international cryptology conference* (1992), Springer, pp. 139–147.

[40] Ellison, C., and Schneier, B. Ten risks of pki: What you're not being told about public key infrastructure. *Comput Secur J 16*, 1 (2000), 1–7.

[41] Espiner, T. Trustwave sold root certificate for surveillance. News article, Feb. 2012. `https://www.zdnet.com/article/trustwave-sold-root-certificate-for-surveillance/`.

[42] Estrada-Jiménez, J., Parra-Arnau, J., Rodríguez-Hoyos, A., and Forné, J. Online advertising: Analysis of privacy threats and protection approaches. *Computer Communications 100* (2017), 32–51.

[43] *Evernym*. `https://www.evernym.com/`.

[44] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. Hypertext transfer protocol – http/1.1. Rfc 2616, IETF, June 1999. `https://www.ietf.org/rfc/rfc2616.txt`.

[45] Fromknecht, C., Velicanu, D., and Yakoubov, S. A decentralized public key infrastructure with identity retention. *Cryptology ePrint Archive* (2014).

[46] Godik, S., and Moses, T. Oasis extensible access control markup language (xacml). *OASIS Committee Secification cs-xacml-specification-1.0* (2002).

[47] Gribneau, C., Prorock, M., Steele, O., Terbu, O., Xu, M., and Zagidulin, D. did:web method specification. W3c internal document, W3C, Dec. 2021. `https://w3c-ccg.github.io/did-method-web/`.

[48] Groth, P., Gibson, A., and Velterop, J. The anatomy of a nanopublication. *Information Services & Use 30*, 1-2 (2010), 51–56.

[49] Grüner, A., Mühle, A., and Meinel, C. Atib: Design and evaluation of an architecture for brokered self-sovereign identity integration and trust-enhancing attribute aggregation for service provider. *IEEE Access 9* (2021), 138553–138570.

[50] Hardt, D. The oauth 2.0 authorization framework. Rfc 6749, IETF, Oct. 2012. `https://www.ietf.org/rfc/rfc6749.txt`.

[51] Hu, V. C., Ferraiolo, D., Kuhn, D. R., et al. *Assessment of access control systems*. US Department of Commerce, National Institute of Standards and Technology, 2006.

[52] Hughes, A., Sporny, M., and Reed, D. A primer for decentralized identifiers. Draft community group report, W3C, Nov. 2021. `https://w3c-ccg.github.io/did-primer/`.

[53] *Hyperledger Foundation*. `https://www.hyperledger.org/`.

[54] *InnoValor: Persoonlijke data, onder controle?* `https://innovalor.nl/Blogs/personal-data-store`, 2016.

[55] *Internet Identity Workshop (IIW)*. `https://internetidentityworkshop.com/`.

[56] Isaak, J., and Hanna, M. J. User data privacy: Facebook, cambridge analytica, and privacy protection. *Computer 51*, 8 (2018), 56–59.

[57] Jones, M., Bradley, J., and Sakimura, N. Json web signature (jws). Rfc 7515, IETF, May 2015. `https://www.ietf.org/rfc/rfc7515.txt`.

[58] Jones, M., Bradley, J., and Sakimura, N. Json web token (jwt). Rfc 7519, IETF, May 2015. `https://www.ietf.org/rfc/rfc7519.txt`.

[59] Jordan, K., Hauser, J., and Foster, S. The augmented social network: Building identity and trust into the next-generation internet. *First Monday* (2003).

[60] Jøsang, A., and Pope, S. User centric identity management. In *AusCERT Asia Pacific information technology security conference* (2005), vol. 77, Citeseer.

[61] Katz, J., and Lindell, Y. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[62] Khan, B. U. I., Olanrewaju, R. F., Baba, A. M., Langoo, A. A., and Assad, S. A compendious study of online payment systems: Past developments,

present impact, and future considerations. *International journal of advanced computer science and applications 8*, 5 (2017).

[63] Khateev, N., and Curran, S. Aries rfc 0454: Present proof protocol 2.0. Aries rfc, Hyperledger, Apr. 2021. https://github.com/hyperledger/aries-rfcs/blob/main/features/0454-present-proof-v2/README.md.

[64] Khovratovich, D. Anonymous credentials. Tech. rep., Hyperledger, June 2016. https://github.com/hyperledger-archives/indy-anoncreds/blob/master/docs/anoncred-usecase1.pdf.

[65] Kilt white paper. Online white paper, BOTLabs GmbH, Berlin, Jan. 2020. https://www.kilt.io/wp-content/uploads/2020/01/KILT-White-Paper-v2020-Jan-15.pdf.

[66] Knublauch, H., and Kontokostas, D. Shapes constraint language (shacl). W3c recommendation, W3C, July 2017. https://www.w3.org/TR/shacl/.

[67] Krämer, N. C., and Schäwel, J. Mastering the challenge of balancing self-disclosure and privacy in social media. *Current opinion in psychology 31* (2020), 67–71.

[68] Laatikainen, G., Kolehmainen, T., and Abrahamsson, P. Self-sovereign identity ecosystems: benefits and challenges. In *Scandinavian Conference on Information Systems* (2021), Association for Information Systems.

[69] Lamport, L., Shostak, R., and Pease, M. The byzantine generals problem. In *Concurrency: the works of leslie lamport*. 2019, pp. 203–226.

[70] Leach, P., Mealling, M., and Salz, R. A universally unique identifier (uuid) urn namespace. Rfc 4122, IETF, July 2005. https://www.ietf.org/rfc/rfc4122.txt.

[71] Lewenberg, Y., Sompolinsky, Y., and Zohar, A. Inclusive block chain protocols. In *International Conference on Financial Cryptography and Data Security* (2015), Springer, pp. 528–547.

[72] Lodder, M., and Hardman, D. Sovrin did method specification. W3c editor's draft, W3C, Feb. 2022. https://sovrin-foundation.github.io/sovrin/spec/did-method-spec-template.html.

[73] Lodder, M., Looker, T., and Whitehead, A. Bbs+ signatures. Draft, MATTR. https://mattrglobal.github.io/bbs-signatures-spec/.

[74] Longley, D., and Sporny, M. Data integrity 1.0. Draft community group report, W3C, Mar. 2022. https://w3c-ccg.github.io/data-integrity-spec/.

[75] Longley, D., Zagidulin, D., and Sporny, M. The did:key method v0.7. W3c unofficial draft, W3C, Mar. 2022. https://w3c-ccg.github.io/did-method-key/.

[76] Lundkvist, C., Heck, R., Torstensson, J., Mitton, Z., and Sena, M. *UPORT: A PLATFORM FOR SELF-SOVEREIGN IDENTITY*. https://blockchainlab.com/pdf/uPort_whitepaper_DRAFT20161020.pdf, 2016.

[77] Maesa, D. D. F., Mori, P., and Ricci, L. A blockchain based approach for the definition of auditable access control systems. *Computers & Security 84* (2019), 93–119.

[78] Mallik, A. Man-in-the-middle-attack: Understanding in simple words. *Cyberspace: Jurnal Pendidikan Teknologi Informasi 2*, 2 (2019), 109–134.

[79] Mertens, W., and Rosemann, M. Digital identity 3.0: the platform for people.

[80] Meyer, U., and Wetzel, S. A man-in-the-middle attack on umts. In *Proceedings of the 3rd ACM workshop on Wireless security* (2004), pp. 90–97.

[81] Mühle, A., Grüner, A., Gayvoronskaya, T., and Meinel, C. A survey on essential components of a self-sovereign identity. *Computer Science Review 30* (2018), 80–86.

[82] Myers, M., Ankney, R., Malpani, A., Galperin, S., and Adams, C. X. 509 internet public key infrastructure online certificate status protocol-ocsp.

[83] Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review* (2008), 21260.

[84] Natarajan, H., Krause, S., and Gradstein, H. Distributed ledger technology and blockchain.

[85] Neuman, B. C., and Ts'o, T. Kerberos: An authentication service for computer networks. *IEEE Communications magazine 32*, 9 (1994), 33–38.

[86] Pandey, N., Pal, A., et al. Impact of digital surge during covid-19 pandemic: A viewpoint on research and practice. *International journal of information management 55* (2020), 102171.

[87] Perlman, R. An overview of pki trust models. *IEEE Network 13*, 6 (1999), 38–43.

[88] Popov, S. The tangle. *White paper 1*, 3 (2018).

[89] Prins, J. R., and Cybercrime, B. U. Diginotar certificate authority breach "operation black tulip". *Fox-IT, November* (2011), 18.

[90] Ragouzis, N., Hughes, J., Philpott, R., Maler, E., Madsen, P., and Scavo, T. Security assertion markup language (saml) v2.0 technical overview. Committee draft 02, OASIS, Mar. 2018. `http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html`.

[91] *Rebooting Web-of-Trust*. `https://www.weboftrust.info/`.

[92] Reed, D., Law, J., and Hardman, D. The technical foundations of sovrin. *The Technical Foundations of Sovrin* (2016).

[93] Rose, J., Rehse, O., and Röber, B. The value of our digital identity. *Boston Cons. Gr* (2012).

[94] Sabadello, M. A universal resolver for self-sovereign identifiers. *Medium*.

[95] Sabadello, M. did:dns method specification. Draft of a potential specification, Danube Tech, Jan. 2022. `https://danubetech.github.io/did-method-dns/`.

[96] Sadq, Z. M., Sabir, H. N., and Saeed, V. S. H. Analyzing the amazon success strategies. *Journal of process management and new technologies 6*, 4 (2018).

[97] Saint-Andre, P., and Klensin, J. Uniform resource names (urns). Rfc 8141, IETF, Apr. 2017. `https://www.ietf.org/rfc/rfc8141.txt`.

[98] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and Mortimore, C. Openid connect core 1.0. Final specification, Nov. 2014. `https://openid.net/specs/openid-connect-core-1_0.html`.

[99] Sankar, L. S., Sindhu, M., and Sethumadhavan, M. Survey of consensus protocols on blockchain applications. In *2017 4th international conference on advanced computing and communication systems (ICACCS)* (2017), IEEE, pp. 1–5.

[100] Schreiber, G., and Raimond, Y. Rdf 1.1 primer. Tech. rep., W3C. `https://www.w3.org/TR/rdf11-primer/`.

[101] Shim, S. S., Bhalla, G., and Pendyala, V. Federated identity management. *Computer 38*, 12 (2005), 120–122.

[102] Smith, B. Ontology. In *The furniture of the world*. Brill, 2012, pp. 47–68.

[103] Soltani, R., Nguyen, U. T., and An, A. A survey of self-sovereign identity ecosystem. *Security and Communication Networks 2021* (2021).

[104] Sporny, M., Guy, A., Sabadello, M., and Reed, D. Decentralized identifiers (dids) v1.0 core architecture, data model, and representations. Proposed recommendation, W3C, Aug. 2021. https://www.w3.org/TR/did-core/.

[105] Sporny, M., Noble, G., Longley, D., Burnett, D. C., Zundel, B., and Hartog, K. D. Verifiable credentials data model v1.1. Recommendation, W3C, Mar. 2022. https://www.w3.org/TR/vc-data-model/.

[106] Sporny, M., and Reed, D. Did primer. Tech. rep., Rebooting Web of Trust. https://github.com/WebOfTrustInfo/rwot5-boston/blob/master/topics-and-advance-readings/did-primer.md.

[107] *Statista: Media usage in an internet minute as of August 2021.* https://www.statista.com/statistics/195140/new-user-generated-content-uploaded-by-users-per-minute/.

[108] *Statista: Number of social media users worldwide.* https://www.statista.com/statistics/278414/number-of-worldwide-social-network-users/.

[109] Steele, O., and Sporny, M. Did specification registries - the interoperability registry for decentralized identifiers. Proposed recommendation, W3C Working Group Note, Nov. 2021. https://www.w3.org/TR/did-spec-registries/#did-methods.

[110] Sunyaev, A. Distributed ledger technology. In *Internet Computing*. Springer, 2020, pp. 265–299.

[111] *The Decentralized Identity Foundation (DIF).* https://identity.foundation/.

[112] *The Linux Foundation.* https://linuxfoundation.org/.

[113] The W3C SPARQL Working Group. Sparql 1.1 overview. W3c recommendation, W3C. https://www.w3.org/TR/sparql11-overview/.

[114] *The World Wide Web Consortium (W3C).* https://www.w3.org/.

[115] Tobin, A., and Reed, D. The inevitable rise of self-sovereign identity. *The Sovrin Foundation 29*, 2016 (2016), 18.

[116] *Trust-over-IP (ToIP) Foundation.* https://trustoverip.org/.

[117] Van der Meulen, N. Diginotar: Dissecting the first dutch digital disaster. *Journal of strategic security 6*, 2 (2013), 46–58.

[118] Veramo. Ethr did method specification. Github specification, Veramo, Nov. 2021. https://github.com/decentralized-identity/ethr-did-resolver/blob/master/doc/did-method-spec.md.

[119] *W3C DID Working Group*. https://www.w3.org/2019/did-wg/.

[120] *W3C Verifiable Credentials Working Group*. https://www.w3.org/2017/vc/WG/.

[121] Werbrouck, J., Taelman, R., Verborgh, R., Pauwels, P., Beetz, J., and Mannens, E. Pattern-based access control in a decentralised collaboration environment. In *8th Linked Data in Architecture and Construction Workshop* (2020), vol. 2636, pp. 118–131.

[122] Whitten, A., and Tygar, J. D. Usability of security: A case study. Tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1998.

[123] Whitten, A., and Tygar, J. D. Why johnny can't encrypt: A usability evaluation of pgp 5.0. In *USENIX security symposium* (1999), vol. 348, pp. 169–184.

[124] Wittmann, L. *Medium Article: Mit der ID-Wallet kannst Du alles und jeder sein, außer Du musst Dich ausweisen.* https://lilithwittmann.medium.com/mit-der-id-wallet-kannst-du-alles-und-jeder-sein-au%C3%9Fer-du-musst-dich-ausweisen-829293739fa0, 2021.

[125] Xu, X., Weber, I., Staples, M., Zhu, L., Bosch, J., Bass, L., Pautasso, C., and Rimba, P. A taxonomy of blockchain-based systems for architecture design. In *2017 IEEE international conference on software architecture (ICSA)* (2017), IEEE, pp. 243–252.

[126] Yeow, K., Gani, A., Ahmad, R. W., Rodrigues, J. J., and Ko, K. Decentralized consensus for edge-centric internet of things: A review, taxonomy, and research issues. *IEEE Access 6* (2017), 1513–1524.

[127] Young, K. Verifiable credentials flavors explained. Tech. rep., COVID-19 Credentials Initiative, Feb. 2021. https://www.lfph.io/wp-content/uploads/2021/02/Verifiable-Credentials-Flavors-Explained.pdf.

[128] Zimmermann, P. R. *The official PGP user's guide*. MIT press, 1995.

Erklärung

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, 27.Juli 2022 Vasil Papanchev