Dynamic Delegation Access Control Protocol for Sharing Verifiable Credentials

P V G S Thamuditha
2025

Dynamic Delegation Access Control Protocol For Sharing Verifiable Credentials

P V G S Thamuditha
Index No: 20001835

Supervisor: Prof T N K De Zoysa

<May 2025>

Submitted in partial fulfillment of the requirements of the
B.Sc. (Honours) in Computer Science Final Year Project

Declaration

I certify that this dissertation does not incorporate, without acknowledgement, any material previously submitted for a degree or diploma in any university, and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, to be made available for photocopying and for interlibrary loans, and for the title and abstract to be made available to outside organizations.

Candidate Name : P V G S Thamuditha

Signature of Candidate **Date:** 2025-06-29

This is to certify that this dissertation is based on the work of Mr. P V G S Thamuditha

Supervisor Name : Prof T N K De Zoysa

Signature of Supervisor **Date:** 2025/06/28

Co-supervisor Name : Dr A P Sayakkara

Signature of Co-supervisor **Date:** 2025-06-29

Contents

List of Figures

List of Tables

List of Listings

List of Acronyms

IdM Identity Management

CIM Centralized Identity Management

FIM Federated Identity Management

IdP Identity Provider

SPs Service Providers

OIDC OpenID Connect

OIDC4VC OpenID Connect for Verifiable Credentials

SSI Self-Sovereign Identity

VC Verifiable Credential

DID Decentralized Identifiers

W3C World Wide Web Consortium

3BI-ECC Three Blockchains Identity Management with Elliptic Curve Cryptography

IRMA I Reveal My Attributes

ABC Attribute-Based Credentials

DLT Distributed Ledger Technology

DHT Decentralized Hash Tables

API Application Programming Interface

ZKP Zero-Knowledge Proof

ADC Access Delegation Credential

DC Delegated Credential

W3C World Wide Web Consortium

VP Verifiable Presentation

IOT Internet of Things

ADC Access Delegation Credential

DC Delegated Credential

DR Delegation Request

ABAC Attribute-Based Access Control

ACA Aries Cloud Agent

OPA Open Policy Agent

ReBAC Relationship-Based Access Control

Abstract

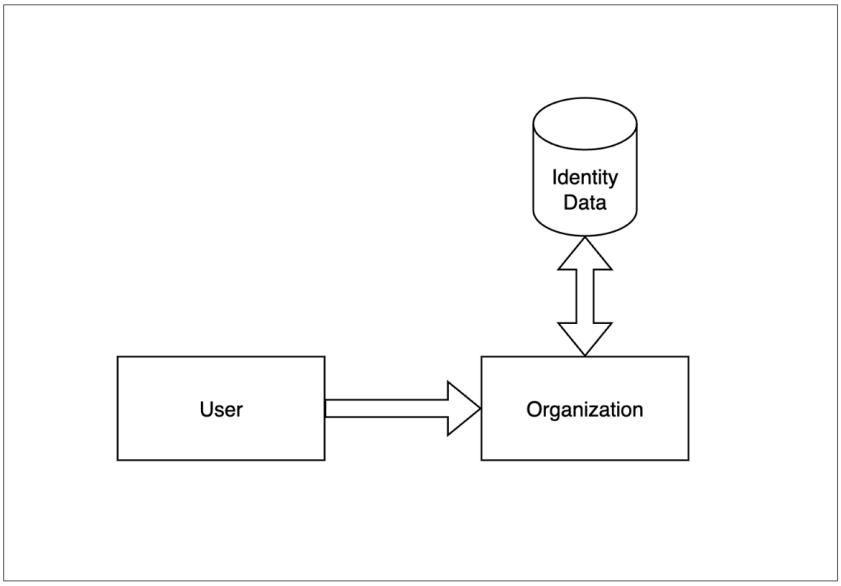
The emergence of Self-Sovereign Identity (SSI) has introduced a paradigm shift in digital identity management, moving back the control over their credentials to their owners. However, current SSI implementations do not present an effective mechanism for access delegation for a Verifiable Credential (VC) without compromising the subject's privacy or control. The objective of this project is a novel Dynamic Delegation Access Control Protocol to address these limitations of present methods, enabling controlled and privacy-preserving credential sharing among participating roles. The Access Delegation Credential (ADC) introduced in this protocol is a verifiable credential issued by the delegator to assert the delegatee's access to the Delegated Credential (DC). Since the original credential is not transferred to the delegatee and remains with the delegator until it is presented to the verifier, this design is preserving the SSI principle while achieving the delegation. The protocol was evaluated under privacy-by-design requirements of SSI. The prototype implemented for the evaluation uses Veramo, Open Policy Agent, with multiple DID methods, including two real-world use cases: student-supervisor access to a library system and employee access using organizational credentials. Performance of the prototype was benchmarked using different key types and DID methods. The protocol achieves the objective of the project, incurring additional overhead compared to traditional delegation methods, but it significantly enhances transparency, enables fine-grained access control, and privacy compliance. It establishes a strong foundation for privacy-preserving delegation in SSI systems.

1 Introduction

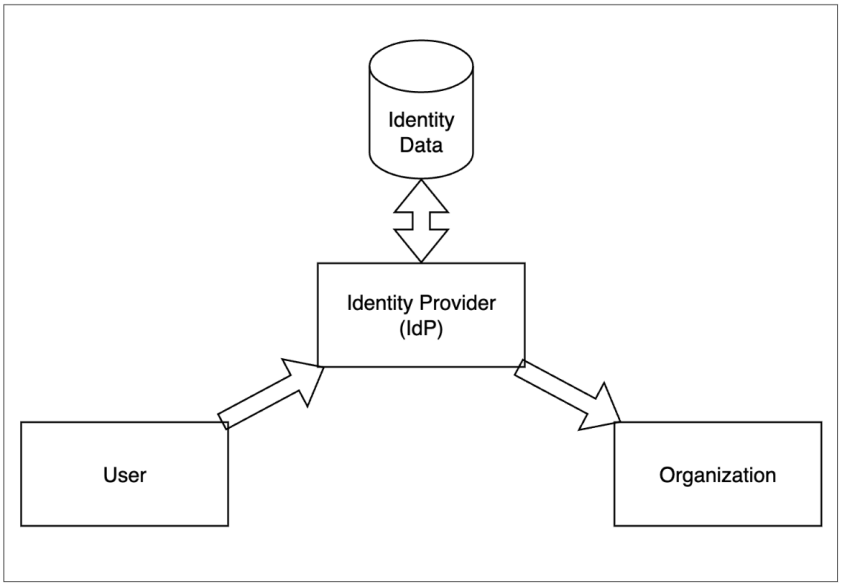
1.1 Background

VC standard is a core element in emerging SSI solutions and even in OpenID Connect for Verifiable Credentials (OIDC4VC), which is an extension of OpenID Connect (OIDC) protocol [7]. It is important to address the current limitations and weaknesses of VC for successful implementations of Identity Management (IdM) solutions. This thesis focuses on the ability to share access to one entity's credentials with another entity. It is important to identify the context where these VCs are being used for analyzing the requirements related to the delegation of access control for sharing VCs.

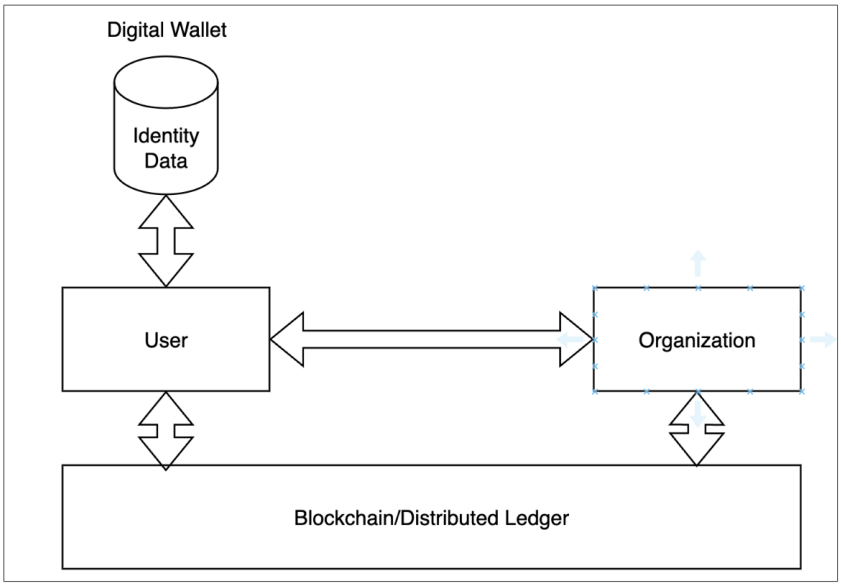
According to [1], digital identity is a vital component of any successful operation on any digital platform. Thus, the concept of IdM has made its way to the digital world. With the evolving requirements of managing digital identities, the concept of IdM has evolved to address these requirements. Mainly, three IdM models were introduced and developed in the following order:



Centralized Identity Management



Federated Identity Management



Self-Sovereign Identity Management

Figure 1: IdM models [1]

Each identity model was introduced to address the shortcomings of the previous models. The next few subsections contain brief introductions on each of the IdM models to cover the identity requirements discussed in each model.

1.1.1 Centralized Identity Management (CIM)

Centralized Identity Management (CIM) is the most mature and simple model out of the three IdM models. Here, the digital identities of entities are managed by a single organization, putting the organization at the center of the ecosystem. CIM systems are compelling concerning security and performance. But this model also presents major challenges like limited portability and resource intensity. Organization at the center also acts as the single point of failure, thus ensuring availability and robustness is a paramount task. The following are the major reasons for moving out of the centralized identity management model:

1. Organizations have to invest in their own centralized management systems because they are disconnected.

2. Users must memorize many login credentials for each system they need to access.

1.1.2 Federated Identity Management (FIM)

Shortcomings of the CIM gave birth to the concept of Federated Identity Management (FIM). There are several definitions given that describe the concept of FIM. [5] describes Federated Identity as the aggregation of identities of the users spanned across the digital space, while [8] is a concept that allows the cooperation of identity processing together with policies and technologies across organizations. From these definitions, we can derive that the basic concept of federated identity management is to allow users to get access to resources across several Service Providers (SPs) using a single identity. The organization that manages the user's identity is called the Identity Provider (IdP). The IdP acts as the intermediary between the users and the SPs managing users' identity and credentials on behalf of the user and authenticating the user on behalf of SPs.

Among the protocols related to FIM, the most promising and widely used ones are the OAuth 2.0 and OIDC [9]. The primary purpose of OAuth 2.0 is the user's authorization, while OIDC, an extended protocol from OAuth 2.0, is used to authenticate the user. Figure 2 illustrates an overview of the OIDC protocol.

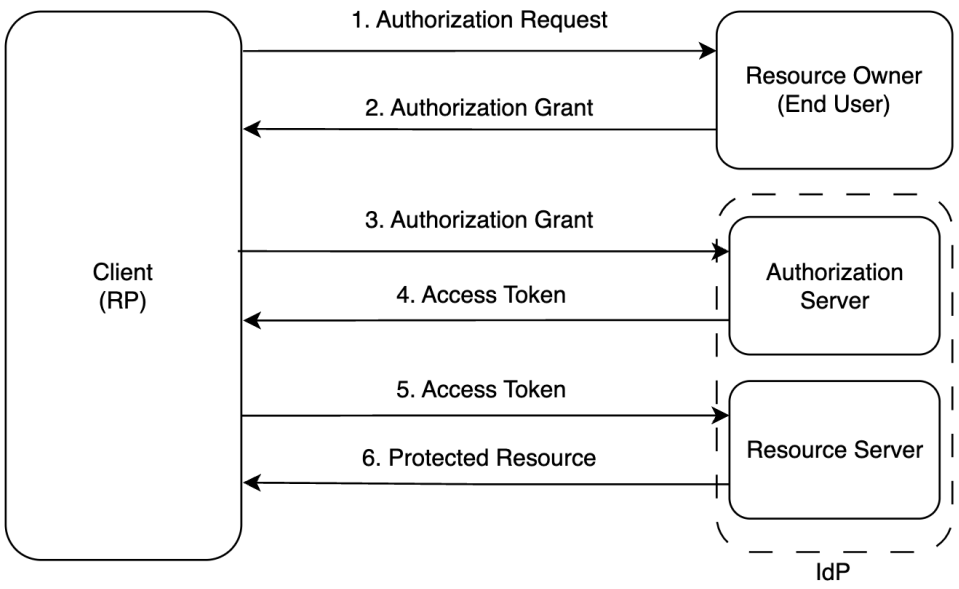


Figure 2: OIDC Protocol Overview

FIM also has several weaknesses:

1. User has no control over the user's credentials
2. Major privacy concerns on the way users' credentials are used
3. Single point of failure at IdP (if IdP fails the whole ecosystem fails)

1.1.3 Self-Sovereign Identity (SSI)

SSI is the most recent identity management model, which is built around the user, giving back control of users' credentials to the users. SSI model replaces previous identity models where instead of an IdP, users themselves present their credentials to a third party where no intermediary is involved in the process [10]. Currently, there are many research studies and standards developed around the SSI model. VCs and DIDs are the most widely used standards that are found across the majority of SSI implementations.

In the SSI model, credentials of the user are stored with the user, either in a digital wallet or any other trusted storage medium. Unlike other IdM models, the user is both subject and holder of their credentials. An ideal implementation of SSI prevents the sharing of users' credentials without the consent of the user. These credentials are issued by a trusted entity, which can be an organization or even the government. When the user's credentials are presented to the organization. The organization must have a way to verify the credentials, and there should be trust established that the credential is issued

by a trusted party. These requirements are satisfied by the VCs and DIDs, which are explained in Sections 1.1.4 and 1.1.5. Figure 3 shows the ontology of the primary SSI concepts and how VCs involve in a SSI system

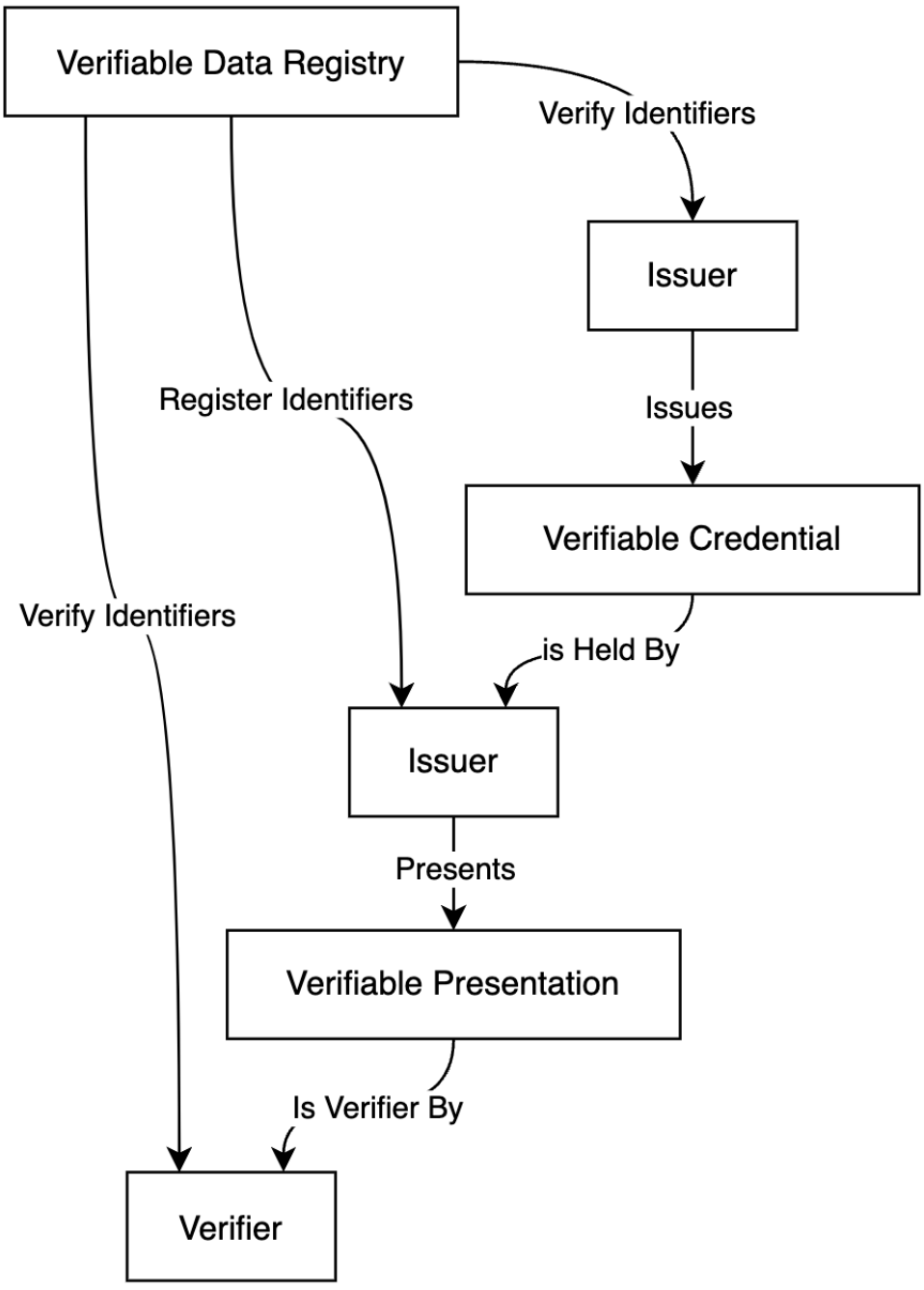


Figure 3: Ontology Chart of Primary SSI Concepts from [2]

1.1.4 Verifiable Credentials (VCs)

To understand the VCs, an understanding of claims and credentials is vital. **Claim** is an assertion or a statement that something is the case. **Credential** is the aggregation of one or more claims about an entity. A **Verifiable Credential** is a collection of one or

more claims that are tamper-resistant, where they assert a certain *subject* which is issued by an *issuer*. Figure 4 represents the roles, responsibilities, and information flows of the basic VC model.

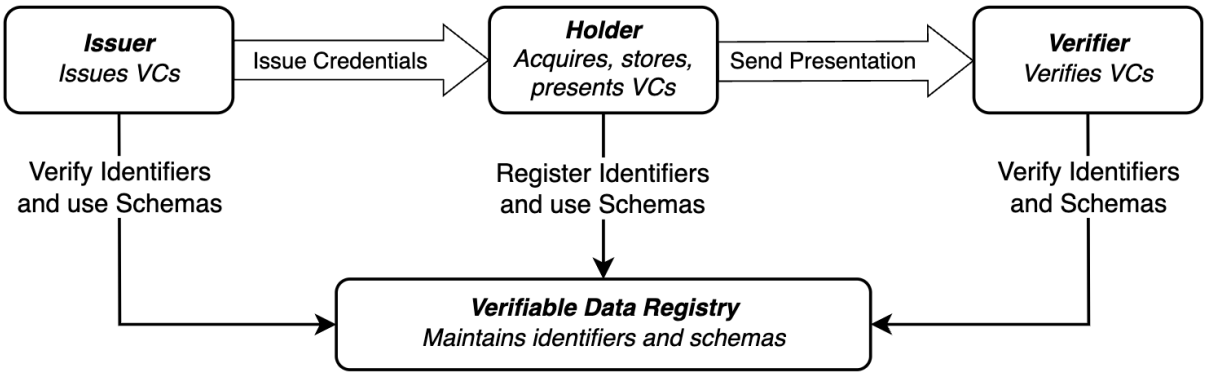


Figure 4: Basic Verifiable Credentials model [3]

Motivated by SSI, VCs were developed and maintained by World Wide Web Consortium (W3C) to support decentralization and enforce users' control over their credentials. VCs also supported by the DIDs, which has lead to wide usage among many SSI implementations. The validity of a VC can be verified using the credential itself, without the involvement of a third party [11]. VCs are made by making credentials tamper-resistant by attaching a proof that verifies the issuance by a trusted issuer. This proof follows the asymmetric cryptographic mechanism. The listing 1 contains a set of claims about the "did:example:student". These claims should be asserted by the university in which the student has enrolled. Let's say the DID of the university is "did:example:university" and there is a DID document (which will be explained in Section 1.1.5), contains a public key of the private key which the university will be using when issuing the VC from these set of claims.

```
{
    "id": "did:example:student",
    "name": "subject's name",
    "course": "Computer Science",
    "university": "University of Colombo School of Computing",
    "enteredDate": "12-06-2020"
}
```

Listing 1: Set of Claims About the Student

The listing 2 shows the VC created by the university by signing and adding relevant metadata to the credential. The highlighted content are added by the issuer. "context",

"type", "issuer", "issuanceDate", and "proof" are metadata attached to the claims so it can be verified. The "context" key contains JSON-LD content or links pointing to JSON-LD files that describe the format of the rest of the credential. "type" should include "VerifiableCredential" and any other type that is defined by the issuer. Both "issuer" and "issuanceDate" contain metadata about the issuance. The "issuer" is important because most of the time it contains a DID that points to the DID document that can be used to retrieve the public key to verify the VC. The "proof" contains the rest of the data that are useful in the verification process of the VC.

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    // rest of the context, if any
  ],
  "type": ["VerifiableCredential", "UniversityCredential"],
  "issuer": "did:example:university",
  "issuanceDate": "2024-12-28T00:00:00Z",
  "credentialSubject": {
    "id": "did:example:student",
    "name": "subject's name",
    "course": "Computer Science",
    "university": "University of Colombo School of Computing",
    "enteredDate": "12-06-2020"
  },
  "proof": {
    "type": "JwtProof2020",
    "jwt":
    ↪  "eyJhbGciOiJFUzI1NksiLCJ0eXAiOiJKV1QifQ.eyJ2YyI6eyJAY29udGV4dC.......",
    "proofPurpose": "verification",
    "verificationMethod": "did:example:university#key-1"
  }
}
```

Listing 2: Signed Credential

In addition to the "jwt" proof format, there are multiple proof formats that can be used with VCs. There is also the option of using Zero-Knowledge Proofs (ZKPs) instead of including sensitive information like instead as the student registration number; the university can issue the attribute 'isStudent'. There are techniques called **selective disclosure**, which allow the holder of the credentials to derive a credential only including a subset of claims in a VC. Still, some implementations of these selective disclosure techniques are at their novel stage.

When the holder presents their credentials to an organization, the holder can create a special type of presentation called Verifiable Presentations (VPs) which includes several VCs and the presentation itself is verifiable. VP also has the same fields "@context", "type" which should include the value "VerifiablePresentation", and "proof". The difference is, instead of an "issuer", VPs has "holder". And there is no credential subject, but the field with the key "verifiableCredential" is an array of VCs of the holder. The listing 3 is an example VP.

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    // rest of the context, if any
  ],
  "type": ["VerifiablePresentation", "ExamplePresentation"],
  "holder": "did:example:holder",
  "issuanceDate": "2024-12-28T00:00:00Z",
  "verifiableCredential": [
    // set of verifiable credentials
  ],
  "proof": {
    "type": "JwtProof2020",
    "jwt":
    ↪  "eyJhbGciOiJFUzI1NksiLCJ0eXAiOiJKV1QifQ.eyJ2YyI6eyJAY29udGV4dC.......",
    "proofPurpose": "verification",
    "verificationMethod": "did:example:holder#key-1"
  }
}
```

Listing 3: An Example Verifiable Presentation

1.1.5 Decentralized Identifiers (DIDs)

To support decentralization efforts, DIDs were developed. A DID is a unique identifier across all the networks. A DID is made up of 3 components, as shown in Figure 5.

<schema_name>:<method_name>:<method_specific_identifier>

Figure 5: DID Format

The schema name is always "did", and the DID method specifies how the creation of DIDs and DID documents are done, along with specifications for resolving, updating,

and deactivating them [12]. Some of the DID methods support URI components like path, query, and fragment as well [12]. Figure 6 shows how a DID is resolved and related artifacts, roles. The DID URI is contained in the DID itself. The DID subject is the owner of the DID, and the DID controller is in control of the DID document that the DID resolves to.

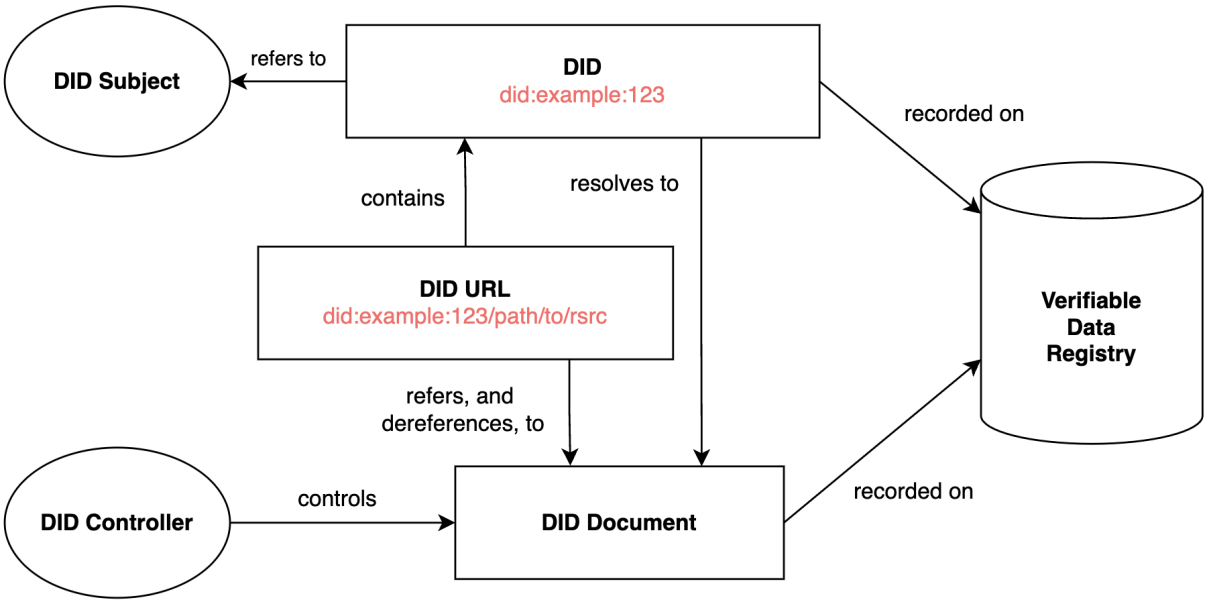


Figure 6: Overview of DID Architecture from [4]

The DID document contains the public information of the DID. The listing 4 is an example DID document. DID controller is the entity that has the authority to change the DID document [12]. Most of the time DID controller is the subject of the VC itself, but it is not always the case. The keys are listed under "verificationMethod", and each key has an "id". When proof is being created, the reference to the key can be attached to the "proof" object in the VC. For the verification, the verifier retrieves this DID document and uses the key referred to in the credential and then verifies using the relevant public key.

```
{
    "@context": [
        "https://www.w3.org/ns/did/v1",
        "https://w3id.org/security/suites/ed25519-2020/v1"
    ],
    "id": "did:example:owner",
    "verificationMethod": [
        {
            "id": "did:example:owner#Key-1",
            "type": "Ed25519VerificationKey2020",
            "controller": "did:example:owner",
            "publicKeyMultibase":
            ↪  "zH3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
        },
        {
            "id": "did:example:owner#Key-2",
            "type": "Ed25519VerificationKey2020",
            "controller": "did:example:owner",
            "publicKeyMultibase":
            ↪  "z9hFgmPVfmBZwRvFEyniQDBkz9LmV7gDEqytWyGZLmDXE"
        }
    ],
    "assertionMethod": [
        "did:example:owner#Key-1"
    ],
    "authentication": [
        "did:example:owner#Key-2"
    ]
}
```

Listing 4: An Example DID Document

The greatest challenge of DID implementation is the wide variety of DID methods available with different implementations, which makes the process of making a universal DID resolver complex. DID methods differ in the ways of creation, resolving, updating, and deleting. Some of the DID methods do not support update and delete operations. The "did:key" is the best example of such a method. Some of the DID methods use blockchains and others do not. DID methods like "did:web" are built on web technologies and does not introduce new technologies. Table 1 contains some of the details on some DID methods. The conclusion is that not all DID methods are readily available and maintained by their creators. Some are not production-ready, but some of them provide SDKs for development by the creators of DID itself. But still, there are considerations to be taken into account when developing an SSI solution incorporating DID methods.

DID Method	Summary
3ID DID Method	Specification of the DID method itself and the ceramic Application Programming Interface (API) it uses is still a draft.
BTC Ordinals DID Method	Utilize the Bitcoin blockchain. For storing an VC, it has a slow transaction speed and high transaction cost.
Key DID Method	Supports several types of keys. The DID specific identifier is the encoded public key. Cannot update the DID document after creation.
GNU Name System DID Method	Only specification published on Tuesday 22nd February, 2022 expired on Thursday 23rd February, 2023.
Trust DID Method	A DID method to be used between trusted parties with a hierarchical order.
Web DID Method	Method-specific identifiers are web addresses that support path, query, and fragment standards. Familiar format of the identifier and use of widely used standards makes it easy to understand and implement.
Binance DID Method	Easy to create a DID. It supports all CRUD operations, and SDKs for development are already available.
Peer DID Method	This supports interactions between trusted parties without a hierarchical order.
Sovrin DID Method	One of the earliest implementations of SSI management that supports all CRUD operations. The needed libraries are readily available.

Table 1: Summary of Some DID methods

1.1.6 Attribute-Based Access Control

There are several access control models that are being used, which are different due to their static and dynamic nature. Unlike traditional access control models like Role-Based Access Control and Access Control Lists, which are static, the Attribute-Based Access Control (ABAC) provides a dynamic approach where access is granted on the evaluation of attributes associated with the user, the resource, and the environment. As oppose to

the Role-Based Access Control model, where the permissions are assigned to static roles, ABAC allows for dynamic and context-aware evaluation.

```
IF user.department == 'computer science'
AND user.year > 1
AND user.status == 'active'
AND resource.classification == 'restricted'
AND env.time >= "08:00" AND env.time <= "22:00"
THEN allow
```

Listing 5: Example ABAC Evaluation

The listing 5 shows an example set of conditions defined on a certain resource. It is equivalent to:

"Active students with more than one academic year from the computer science department can access 'restricted' materials during hours 8 a.m. to 10 p.m.."

ABAC allows fine-grained access control for resources that need to be protected. The main idea behind the ABAC, the user must be someone who can be trusted with the resource, or accessing the resource by a user with these attributes doesn't pose a risk.

1.2 Motivation

IdM is a key component of any digital platform [1]. Along with the evolving requirements, IdM has evolved to address several concerns about the digital identity of users. Among existing IdM models, the SSI model is built focusing on the SSI principle that makes the subject of the credential the absolute owner of that credential, allowing the subject to exercise authority over their credentials freely.

The ecosystem around SSI is still in an early stage and continues to develop, employing different technologies to address issues related to IdM. The VC and DID were developed by the W3C to support the SSI model. Though VCs and DIDs are currently being used in some of the implementations, these implementations do not support the full capabilities that are expected from them [13]. The motivation behind this project is to address the privacy concerns related to the delegation of VC by the subject to another subject and evaluation of the solution.

1.3 Research Gap

According to [6], the current VC model is "Holder-Centric". When the subject wants to share the subject's credentials, VC provides the option of delegation. The VCs allow two types of delegations. The first one is the delegation of the ability to create a verifiable presentation on behalf of the user, by specifying the delegate as a controller. The second method is the attenuated delegation method.

1.3.1 Adding The Delegatee As a Controller

This delegation method is a powerful method that allows the delegatee to act on behalf of the delegator. Listing 6 is an example did document where the owner of the DID document, "did:example:delegator" has delegated access to his DID by adding "did:example:delegatee" as another controller of that DID. The highlighted lines of the listing 6 show the verification methods available for "did:example:delegator" DID. As long as the delegatee's public key is listed as a verification method, the delegatee can use the delegator's DID and VCs to create VPs and present them to a verifier.

```
{
    "@context": [
        "https://www.w3.org/ns/did/v1",
        "https://w3id.org/security/suites/ed25519-2020/v1"
    ],
    "id": "did:example:delegator",
    "verificationMethod": [
        {
            "id": "did:example:delegator#delegatorKey1",
            "type": "Ed25519VerificationKey2020",
            "controller": "did:example:delegator",
            "publicKeyMultibase":
            ↪  "zH3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
        },
        {
            "id": "did:example:delegatee#delegateeKey1",
            "type": "Ed25519VerificationKey2020",
            "controller": "did:example:delegatee",
            "publicKeyMultibase":
            ↪  "z9hFgmPVfmBZwRvFEyniQDBkz9LmV7gDEqytWyGZLmDXE"
        }
    ],
    "assertionMethod": [
        "did:example:delegator#delegatorKey1",
        "did:example:delegatee#delegateeKey1"
    ]
}
```

Listing 6: Example Delegated DID Document

This is ideal for some use cases, like where a DID of one entity needs to be controlled by another entity. For example, a DID of a child is being controlled by a parent, or DID of a vehicle, or DID of an Internet of Things (IOT) device being controlled by the owner. But for a use case like a supervisor giving access to his credentials for some library system, so a student can access it, this method imposes unwanted authority on the student. A malicious student can exploit this. The owner of the DID document cannot intervene or track when the owner's DID is being used because its usage doesn't involve the owner's mediation. And also the owner does not have the facility to define fine-grained access to the owner's DID or VCs.

Apart from the above security concern, this method is not supported by DID methods like "did:key" [13], which do not support the addition of new keys or DID document updates. So the mechanism used here is not inherently supported by some of the DID

methods available.

1.3.2 Attenuated Delegation

In attenuated delegation, instead of giving access to the owner's DID, the owner shares a credential derived from the original credential either by using ZKP, selective disclosure, or both. There is also the option of creating a new credential, including the claims from the original credential needed without deriving from the original credential.

Selective disclosure mechanisms like BBS+ are developed for the purpose of revealing only the needed claims that are asserted by an issuer. But according to [13], implementations of selective disclosure are incomplete as of today. In addition to its incompleteness, since the owner gives the credential to the delegate, the owner can't intervene when the credential is being used. So, the attenuated delegation method also cannot prevent the holder from misusing the delegated credential, or the owner cannot intervene and know how the owner's credential is being used. And this method violates the SSI principle as well.

Current VC model lacks the facilities to delegate a credential to another without violating the SSI principle, and preventing the misuse of delegation. The "Subject-Centric" model suggested by [6] loses the ability of selective disclosure. VCs are plain text, and any party that encounters them will have the ability to map the relationships between the related parties, and it will also reveal claims of the owner that are not needed for the encountering parties. There should be a well-defined procedure to share access to a certain credential without sharing the credential to parties that need it for verification, and it should support, if not all, at least the majority of DID methods as well.

1.4 Scope

This project aims to improve the basic VC model by establishing a protocol that enables the dynamic delegation of access control that enabling a subject to share the subject's VC without transferring or allowing more access than the holder needs. While preserving the SSI principle, it will broaden the benefits of using VCs as a standard for decentralized IdM. This has the potential to be implemented on the sides of the organizations that aim to implement a decentralized IdM system, allowing management access to resources and authority of job roles. There are three main roles of an SSI ecosystem: the issuer, the holder, and the verifier. Unlike individuals, organizations may fulfill more than one

role in a single ecosystem. Management of these roles needs some requirements which is not currently covered by the VC specifications. This solution can also establish the foundation for addressing this problem.

1.5 Research Questions

1. To what extent can an access control delegation for sharing verifiable credentials be implemented without violating SSI principle?
   Current delegation procedures transfer full control of the subject's credentials to a holder. In both of the delegation methods, the holder can use the other subject's credentials, similar to the subject of the credentials, without any obstacle. This violated the SSI principle. The goal is to design a protocol that can facilitate the delegation of access control that can limit the holder's authority over another subject's VCs.

2. What methodology can enable access control delegation for sharing verifiable credentials in a dynamic context?
   Most SSI solutions have employed Distributed Ledger Technology (DLT) and Decentralized Hash Tables (DHT) to implement SSI solutions. It is crucial to evaluate the availability and performance of the newly designed protocols in a dynamic context because the performance and the requirements of the underlying technologies are different, and their performance under dynamic conditions will affect the success of the protocol.

3. What are the privacy, and performance implications of dynamic access control delegation for sharing verifiable credentials?
   Ensuring the privacy of the user and the potential security vulnerabilities that an attacker can exploit are serious considerations concerning a subject's identity. Evaluation of the protocol should be done under privacy implications, along with the performance implications of the new protocol.

Algorithm Delegation Time Taken Delegation Memory Usage Verification Time Taken Verification Memory Usage Retrieval Time Taken Retrieval Memory Usage Ed25519 3.468516041092540 79.1005993150685 2.810180958902640 79.1472602739726 2.361374369860720 79.1472602739726 Secp256k1 3.510733479445750 81.39768835616440 2.943896095895160 81.48565924657530 2.283267657535650 81.48565924657530

1.6 Research Aim

The research aim of this project is to design a protocol capable of dynamic delegation of access control for sharing verifiable credentials and to evaluate the designed protocol under selected use cases.

1.7 Research Objectives

Objectives to be completed to answer the above research questions are summarized as follows:

- Identify DID methods and the VC model suitable for access control delegation
- Design a protocol for access control delegation with the following features,
  - Selective Disclosure of credentials
  - Revocation of delegations
- Examine the behavior of the designed protocol in a dynamic context
- Qualitative evaluation of privacy of the protocol (based on privacy by design requirements by [13])
- Simulate the protocol under the following use cases and evaluate its performance
  1. Delegation of access for using a library system to access specific documents using a supervisor's subscription
  2. Delegation of access control for VCs of an organization to its employees
- Measure the performance under different key types and their algorithms
  1. Ed25519
  2. Secp256k1
  3. P-256

1.8 Evaluation

1.8.1 Privacy Evaluation

Privacy requirements of the protocol were chosen from the privacy-by-design principles given by [13], which should exist in the SSI solution. They are,

1. **Multi-Show Unlinkability** - Several credential presentations derived from the same original credential and transmitted over several sessions cannot be linked by verifiers

2. **Issue-Show Unlinkability** - Any information collected at the time of credential issuance cannot be used subsequently to establish a link between the credential presentation and the original credential

3. **Non-Correlation** - Non-correlation is the inability of any actor, including curious actors on the infrastructure, for example, or providing a service like DNS servers, to learn about the activity of the holder, by linking their different identifiers together, or by linking an activity to the specific holder

4. **Ability to use selective disclosure techniques**

5. **Transparency** - Transparency is fundamental for privacy and data protection, since it guarantees that the data subject is aware of how their data is being processed and where it is stored, etc

6. **Non-traceability** - Non-traceability means that a curious actor is not able to trace the activities of a certain user back to them

Evaluating the protocol on these privacy goals follows a qualitative assessment. Following the protocol and identifying possible instances where these requirements can be violated will determine whether the protocol adheres to these requirements.

1.8.2 Performance Evaluation

Record the time taken for the following steps of the delegation,

1. Credential delegation

2. Presentation & verification of the delegation

The time taken for each step and its aggregate are recorded and compared.

Evaluation Between Available Methods and Designed Protocol

As described in Section 1.3, there are already two methods that support delegation. They are adding the delegate as a controller and attenuating delegation. The performance of the protocol against these available methods is measured and compared. The

implementation for this evaluation is done through the first use case using a DID method that supports all 3 methods. The comparison of time taken allows us to identify the resulting performance difference between the new protocol and existing methods.

Evaluation Between Types of DID Methods

As per [14], there are different types of DID method implementations available. DID methods use both blockchain-based and blockchain-less implementations. It is important to identify the performance difference across the different kinds of implementations that are available.

The expected comparison is to compare the difference in time taken for the protocol to achieve the same result, under blockchain-based and blockchain-less implementations. The SSI solutions employ both of these types of implementations, and therefore, a mixture of these DID methods is available. The difference in implementation effects is mainly on DID document resolution. In addition to resolution, there is also DID creation, DID document update capabilities as well, but it does not fall under the scope of this project.

Evaluation Between Different Signing and Verification Algorithms

For this evaluation, a DID method that allows several different algorithms, and that does not take time for resolution of the DID document, will be chosen. The reason for selecting a single DID method is that the difference between available DID methods and the latency in retrieving DID document affects the accuracy of the measurements.

More details on implementation will be discussed in Section 4. Careful execution of the use cases will help to identify the performance implications of the designed protocol.

2 Literature Review

To understand IdM and to identify a possible research gap, a preliminary literature review was conducted on the topics of **Identity Management, Federated Identity Management, Self-Sovereign Identity**, **Verifiable Credentials**, **Decentralized Identifiers**, **Dynamic Access Control**, and **Selective Disclosure**.

The scenarios where FIM is applicable in real-life applications are explained in [5]. Sectors like E-health, E-government, E-learning, and E-business considerably benefit from FIM over traditional approaches. Benefits and challenges that exists in FIM ecosystem together and a "Circle of Trust" that exists within FIM implementations (refer to Figure

7) are explained in [5]. With the research of [8], considerable challenges were present in implementations of FIM.

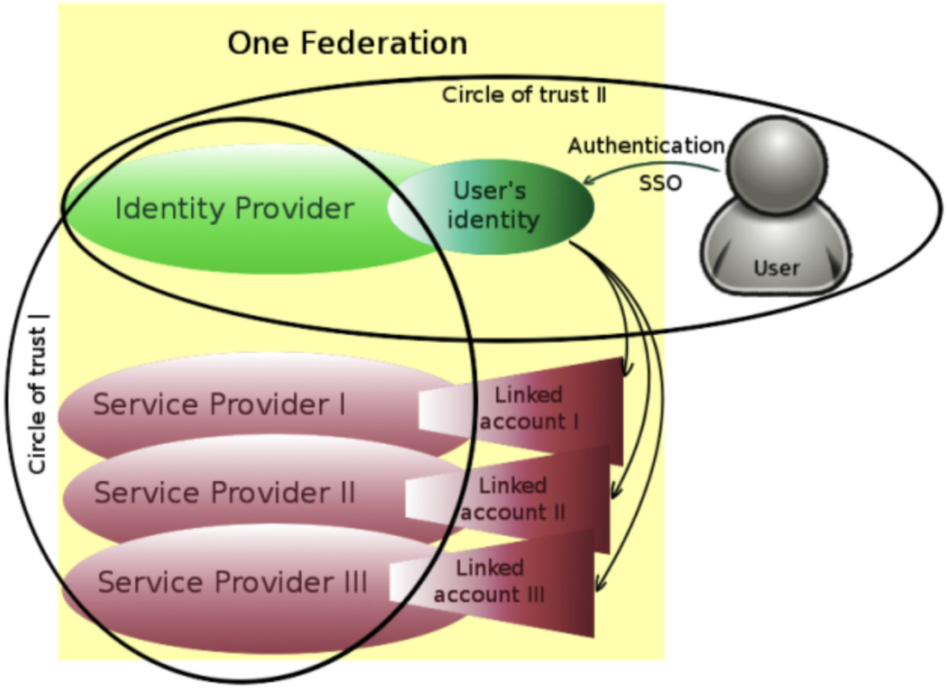


Figure 7: Circle of Trust [5]

The privacy-by-design principles that should be considered when developing an FIM solution are explained in [15]. Their research further explains the architectural requirements and business requirements that relate to these policies. [16] explored the security issues of FIM in cloud computing. Even well-known and widely used FIM protocols like OIDC and OAuth 2.0 possess some privacy-related weaknesses. This gave birth to the concept of SSI.

SSI has gained the attention of the research community during the recent period due to several new technologies like blockchain and standards like DIDs and VCs. [17] specified the use cases of SSI, explaining that an SSI solution must consider all these use cases to create a successful SSI solution. Their research discusses some of the existing SSI solutions by using these use cases as a benchmark. In addition to these use cases, the work of [13] presents 8 privacy requirements that should be present in an SSI solution. These requirements focus on the basic SSI principle and its purpose. The following are the 8 privacy requirements specified by [13] that are desirable for a IdM solution that follows SSI model:

1. Pseudonymity

2. Multi-Show Unlinkability
3. Issue-Show Unlinkability
4. Non-Correlation
5. Selective Disclosure
6. Transparency
7. Non-traceability
8. Confidentiality of Wallet Communication

uPort is one of the earliest SSI implementations, where it utilizes the power of the public permissionless Ethereum blockchain and smart contracts. [18] explained the components of uPort and an overview of the protocol flow. There are 3 types of smart contracts and 4 servers needed to make uPort work. It supports almost all of the use cases specified by [17]. At present, the uPort project is archived and continued as two separate products, Serto and Veramo, both focusing on decentralization, thus continuing the efforts of uPort.

Sovrin is another SSI solution that is explored in the work of [19]. Sovrin works on the permissioned public blockchain Hyperledger. Unlike uPort, only trusted entities known as **stewards** can engage in the consensus protocol. These stewards are governed by the Sovrin Trust Framework. While Sovrin supports both edge agents in the user's mobile device or cloud agent to manage the user's credentials, this protocol is more complex than uPort. But it also supports [17] use cases.

There are several other SSI implementations like Three Blockchains Identity Management with Elliptic Curve Cryptography (3BI-ECC) [20] and ShoCard that rely on blockchain technology. But the Implementation of the I Reveal My Attributes (IRMA) [21] by utilizing Attribute-Based Credentials (ABC) shows that blockchains are not necessary for an SSI solution. Blockchains do provide many of the needed functionalities for decentralization, but are not mandatory to achieve it [22]. [22] further compared these implementations and questioned whether, though there are many implementations and they all offer myriad functionalities, a user wants to use them. This question seems to be unanswered as of right now.

Across all these SSI solutions, two standards have been used in almost every one of them. They are DIDs and VCs. VCs are the primary focus of this research. VC was

developed to support the decentralization approach in IdM. A VC has the same ability as a physical credential to represent the information issued by a certain issuer [3]. Employment of the digital signature technique, VCs, becomes tamper-resistant, thus making it more trustworthy. [23] explains that even though VC is built to enhance privacy, there are 2 privacy-related considerations in VCs. Aggregation of credentials always reveals more about the user than the user intended, and users' usage patterns may lead to unexpected correlation when VCs are presented to the same verifier more than once. [6] describes the basic model of the verifier credentials as a Holder-Centric model. The reason for this is that whoever the subject is, the current holder of the credential can use the credentials without the involvement of the subject. Figure 8 represents this problem.

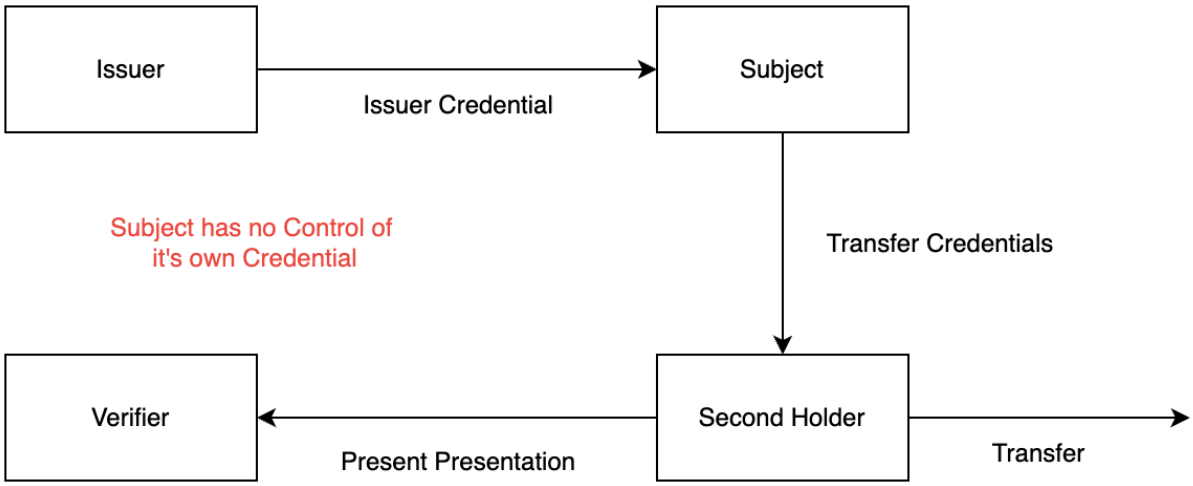


Figure 8: Holder-Centric model [6]

To mitigate this problem, [6] suggests a Subject-Centric model where the transfer of credentials happens after encryption, and at verification, the verifier has to request the decrypted credentials from the subject (Figure 9). [6] refers to this as "Subject-Centric Model". However, this acts as a barrier to selective disclosure.

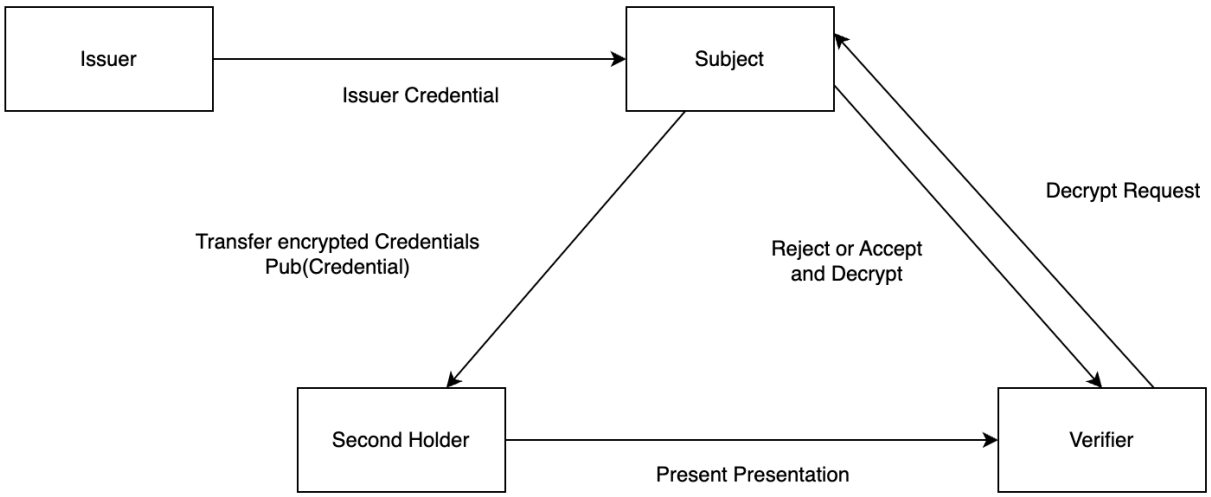


Figure 9: Subject-Centric model [6]

As per W3C's definition, in SSI systems and users exist independently from each other [24]. Though VCs help to achieve this, it is not the complete solution. The way VCs are used plays a huge role in overcoming the weaknesses of the SSI model and any other concept used in the implementation. [11] mentioned requirements that should be satisfied by related technologies for VCs to work. Digital wallets must handle both the management of VCs and cryptographic requirements as well, and the verifiable data registry must be capable of managing DIDs. Verifiable Credentials (VCs) Data Model 2.0 latest draft was published on Thursday 20$^{\text{th}}$ March, 2025.

Work of [7] mentions enhancing OIDC to incorporate VC with the DIDComm protocol. DIDComm protocol operates over protocols like HTTP, and it is based on DIDs to support dynamic peer-to-peer messaging [7]. According to [13] VC break the ledger dependencies of the SSI model. The work of [2] explores the data aggregation management of SSI with VCs and DIDs. It mainly focuses on data aggregation in a decentralized network. Work of [13] includes a survey done by CheckD on most widely used VC schemas:

1. Hyperledger Anoncreds
2. JWT
3. JSON-LD
4. JSON-LD with BBS+

Works of [14] and [12] have done extensive analysis of the capabilities of decentralized technologies related to identity management, especially on VCs and DIDs. [14]

analyzed the CRUD operations that can be done via each DID method. Their survey further extends on classifying all 168 DID methods based on the nature of DID method implementation. [14] characterized the registry of the each DID method on following characteristics,

- Public ledger
- Private ledger
- Permissioned ledger
- Permissionless ledger
- Ledger-agnostic

In addition to this classification work of [14], it covers an extensive survey on available DID methods. [14] covers specification of DID methods categorising them into the above 5 categories, completeness of the specification, and the creation and last updated date of the specification at the time of the survey. According to [14], 28.6% of the DID methods use public blockchains while the majority of those public blockchain-based DID methods use Bitcoin (10.34%) and Ethereum (51.72%). According to [14], a DID document contains six optional fields that are needed according to the need. They are,

1. DID
2. Set of cryptographic material
3. Set of cryptographic protocols
4. Set of service endpoints
5. Timestamps
6. JSON-LD signature

The work of [12] mainly looked at the use cases and current DID and VC implementations, highlighting that the current implementation is still at the early stages. Veramo, a Javascript Framework for managing DIDs and VCs and the Hyperledger Aries blockchain together with the Aries Cloud Agent were analyzed by [12], including them as well developed tools for realizing SSI. [13] list down DID methods supported across SSI vendors as:

1. did:indy/did:sov
2. did:web
3. did:key
4. did:ethr

Work done by [25], [26], and [27] surveys the technology and implementations around VCs, especially on selective disclosure techniques. Their work reveals **atomic credentials**, **hash-based techniques**, **signature-based techniques**, and **zero-knowledge proofs** as most commonly used selective disclosure techniques, and often a combination of them is used. [25] further evaluates the cryptographic algorithms that are being used for VC signing alongside algorithms like BBS+ that are used to reveal part of the credential, on performance when generating and verifying of proofs, complexity, key size, and the size of the proof against the key size. Though there are many selective disclosure techniques available, according to [13], current implementations of JSON-LD signatures do not support selective disclosure. [13] has conducted a comprehensive survey on SSI technologies currently available and listed out most commonly used DID methods and proof formats for VCs. Any solution introduced to the SSI ecosystem must support these widely used DID methods and proof formats.

The review done by [28] on "Access Control for IoT-based Big Data" mentions the currently implemented access control techniques. Dividing them mainly into 2 categories, static and dynamic, as they further explain currently in-place methods that are used in this domain. For dynamic access control, the two main 2 methods used are trust-based access control and risk-based access control. [29] introduces a Relationship-Based Access Control (ReBAC) model and policy language that is suitable for dynamic access control scenarios. The work of [30] covers the ABAC model for web services. [30] also highlights traditional access control models like Role-Based Access Control, User-Based Access Control, and Access Control Lists. In addition to access control strategies, [31] specifies the challenges in access control as:

1. Dynamic Requirements and Automation
2. Complexity and Scalability
3. Equitable Access and Efficient Negotiation
4. Regulatory Compliance and Adaptability (GDPR, CCPA, etc.)

3 Methodology

The roles that participate in the protocol are as follows,

1. **Delegator** - owns the credential and delegates the access
2. **Delegatee** - requests and acquire access to the credential
3. **Verifier** - verify the presented presentation and retrieve the delegated credential

This protocol gives access to another subject's credentials and does not enforce any restrictions on credential exchange between the holder (in this case, the delegatee) and the verifier. Instead, the proof of access can be presented with a set of any other proofs that the verifier needs. This proof of access is another VC called the **ADC**. The listing 7 is the format of an ADC.

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://example.com/contexts/access-delegation-v1"
  ],
  "id": "example-credential-id",
  "type": ["VerifiableCredential", "AccessDelegationCredential"],
  "issuer": "did:example:delegator",
  "issuanceDate": "2024-12-28T00:00:00Z",
  "credentialSubject": {
    "id": "did:example:delegatee",
    "credentialId": "example-delegated-credential-id",
    "attributes": {
      "over-18": true
      // some other claims
    },
    "service": {
      "type": "DIDComm",
      "serviceEndpoint": "https://example.com/credentials"
    }
  },
  "proof": {
    "type": "JwtProof2020",
    "jwt":
    ↪  "eyJhbGciOiJFUzI1NksiLCJ0eXAiOiJKV1QifQ.eyJ2YyI6eyJAY29udGV4dC.......",
    "proofPurpose": "verification",
    "verificationMethod": "did:example:delegator#key-1"
  }
}
```

Listing 7: Example Access Delegation Credential

"type" of the ADC should include both "VerifiableCredental" and "AccessDelegationCredential". As per VC standard, claims about the delegatee is included under the **"credentialSubject"** key. Those claims are,

1. **id** - Id of the delegatee.
2. **credentialId** - Id of the credential owned by the delegator.
3. **attributes** - This is a JSON object that has the claims asserted by the delegator. These claims define the delegatee's access to the DC.
4. **service** - This key defines the service that the verifier should use to obtain the DC. The ADC should be sent as the payload. **"service"** is the common standard used

in DID documents to specify services of the owner of the DID

3.1 Protocol

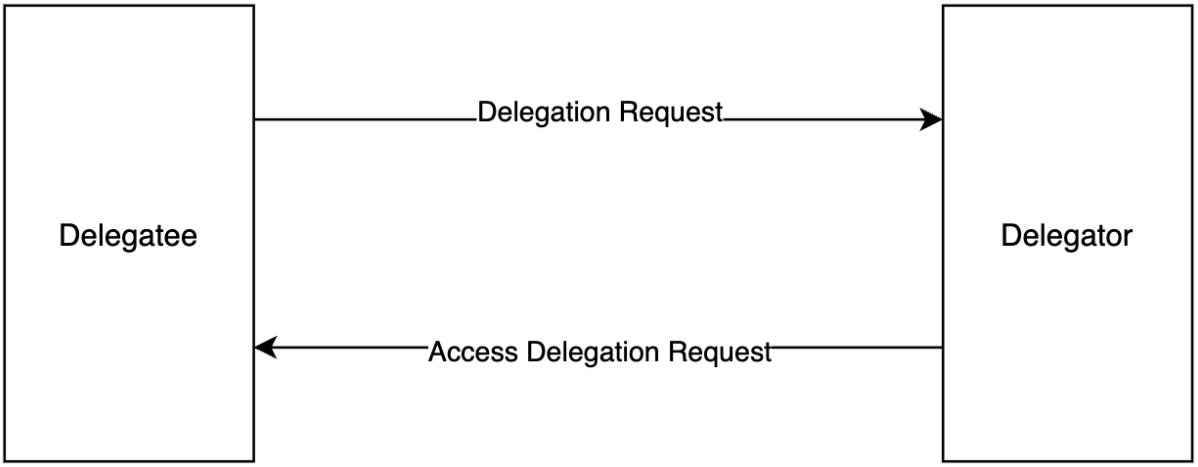


Figure 10: Request Access From Delegator

Figure 11: Send Verifiable Presentation to Verifier

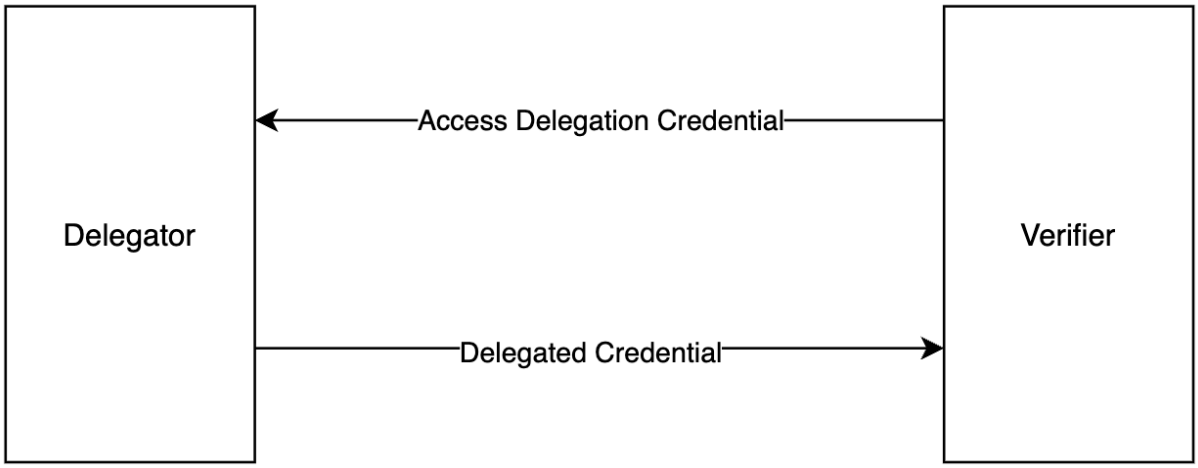


Figure 12: Retrieve Delegated Credential

The figures 10, 11, and 12 show a higher-level overview of the protocol. A detailed breakdown of the protocol is as follows,

1. The delegatee sends the **Delegation Request (DR)** to the delegator. This DR should contain a VP with whatever claims the delegator requests for the specific credential.

2. The delegator verifies the received VP. Then the delegator evaluates the claims provided and creates the **ADC**, including assertions made by the delegator on the delegatee. Created ADC is sent back to the delegatee. If the VP is not valid or the claims provided are not enough, the delegator sends back an appropriate response.

3. The requested claims are sent to the verifier as a VP including the ADC received in step 2.

4. The verifier verifies the VP received. Except for the ADC, other credentials are handled by mechanisms specified by the verifier.

5. The verifier sends the ADC via the **"service"** method specified in the ADC to the delegator (or to the resource specified by the delegator).

6. The delegator first verifies the ADC and then evaluates the access by evaluating claims specified in the ADC against access policies specified on the DC. How access is evaluated is explained in Section 3.2. If ADC contains sufficient claims, the DC is sent back as a response. If either the ADC is not valid or ADC does not contain sufficient claims, an appropriate response is sent back.

7. With the retrieval of DC, the verifier should verify it. After verification, it can be handled like any other credential.

3.2 Access Evaluation

Follows **ABAC** model, since VC are made up of key value pairs, which is equivalent to an attribute identifier and its value as in (1).

$$Attribute \equiv (< Attribute\_Identifier >, < Value >) \tag{1}$$

The ADC contains the attributes of the delegatee asserted by the delegator. Let X be the delegator, then X_{attr} is an attribute of the delegator. Then the set of attributes of the X specified in the ADC can be defined as (2).

$$X_{attr\_set} = \{X_{attr_1}, X_{attr_2}, X_{attr_3}, ...\} \tag{2}$$

Condition is a set of values allowed for a certain attribute. If the attribute value contained in the ADC is an element of the condition, the condition is satisfied. Let Y be the resource (DC) X is trying to access. Then a condition on Y is in the format of (3).

$$Y_{condition} \equiv Attribute\_Identifier < OP >< Value > \tag{3}$$

Where, $< OP >\in \{=, \neq, >, <, \geq, \leq\}$. A condition allows a set of values over the range of the attribute (4).

$$Y_{condition} = \{value_1, value_2, value_3, ...\} \tag{4}$$

and if $X_{attr}[Attribute\_Identifier] \in Y_{condition} \rightarrow true$.

A **policy** on Y is a set of conditions defined on Y (5). For a policy to be satisfied, all conditions of that policy should be satisfied (6).

$$Y_{policy} = \{Y_{condition_1}, Y_{condition_2}, Y_{condition_3}, ...\} \tag{5}$$

$$Y_{policy} \iff Y_{condition_1} \cap Y_{condition_2} \cap Y_{condition_3} \cap ..... \tag{6}$$

A **policy group** on Y is a set of policies defined on Y (7). For accessing Y, X should satisfy at least one of the policies defined in the policy group (8).

$$Y_{policy\_group} = \{Y_{policy_1}, Y_{policy_2}, Y_{policy_3}, ...\} \tag{7}$$

$$Y_{policy\_group} \iff Y_{policy_1} \cup Y_{policy_2} \cup Y_{policy_3} \cup ..... \tag{8}$$

This strategy allows for defining multiple combinations of conditions for accessing a credential. It also keeps the room for fine-grained access control. The owner of the credential (the delegator) can revoke access at any time, by changing a policy revoking access to a group of delegates, or can revoke access to a single delegatee by employing a strategy like **revocation list**.

4 Implementation

For evaluation, the protocol implementation is done by selecting widely used DID methods, proof formats, and other VC related technologies from previous surveys. Figure 13 shows the architecture of the implementation. This implementation includes minimum requirements for the protocol to function. In real-world implementation, requirements may be different, and more considerations should be taken into account in implementing. However, the design of the protocol is made to support widely used VC and DID related technologies. The DID methods and proof formats used in the implementation are chosen from [13], and the availability of Free and Open Source implementation, together with their alignment with evaluation goals.

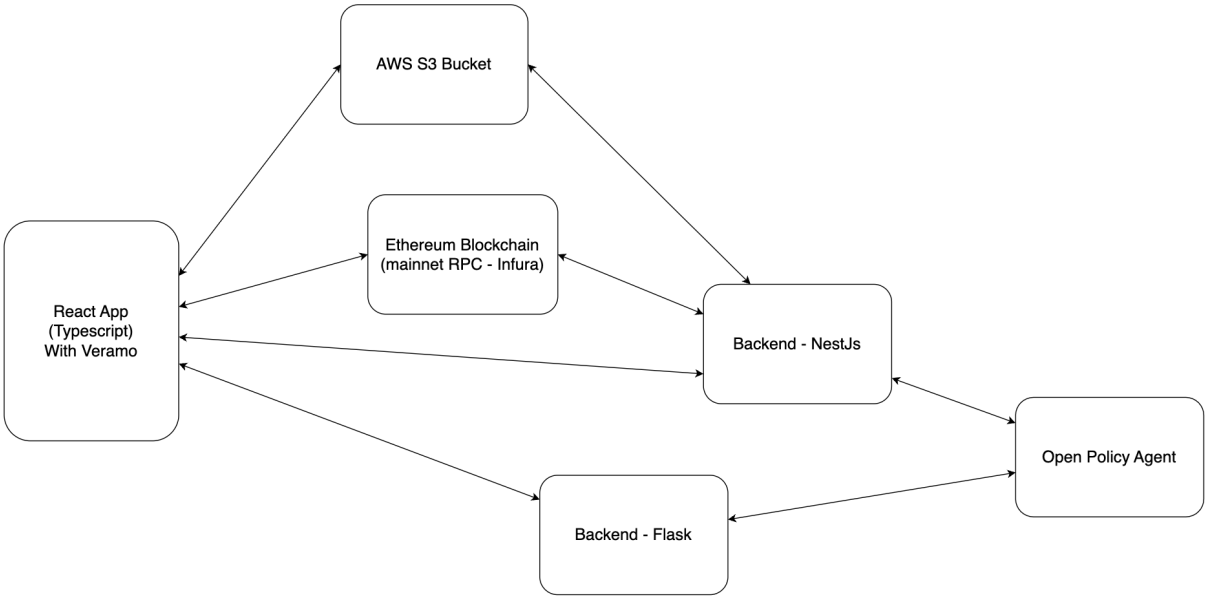


Figure 13: Implementation Architecture

4.1 React App

The React app is the application that initiates the protocol flow and imitates the actions of the delegatee. It is integrated with Veramo, a typescript framework that runs on

a plugin-driven architecture. Veramo runs on Node, making it multi-platform. It was chosen according to the survey done by [12]. In addition to Veramo, the React app is integrated with other necessary libraries like axios, which was used to communicate with NestJs & Flask backends and a local data store for storing DID and related metadata.

Veramo was configured for the use cases 1 & 2, with the ability to support the "did:web" method and the "did:ethr" method. For "did:ethr", the mainnet RPC provided by the Infura project was used due to its compatibility with Veramo. It provides DID document resolution, VC creation & verification, and VP creation & verification. To compare algorithms, the "did"key" method was used by using the algorithms provided by Veramo itself.

4.2 Amazon S3 Bucket

For the "did:web" method, hosting of the DID document should be done on a certified domain. Amazon S3 Bucket provides easy and simple static resource hosting. The DID document is a JSON file, and resolution of the "did:web" points to a URL where the DID document is hosted. Since Veramo has implemented this resolving, the hosting part has to be handled manually, and it was done by integrating **Amazon SDK** into the React app.

4.3 NestJS Server

The purpose of the NestJS server is to imitate the actions of the delegator. The reason for choosing NestJS is that it is a TypeScript-based framework, Veramo library can be used as a plugin. For both use cases, it provides the delegator's function. For this implementation, "serviceEndpoint" in the ADC specifies an endpoint of this server which verifies the ADC and then evaluates the access with the help of OPA. More details on OPA will be discussed in Section 4.5.

4.4 Flask Server

The reason for using another backend server for this implementation is the lack of rich libraries that support multiple signing algorithms. Although Veramo supports both Ed25519 and Secp256k1, to implement a wide variety of algorithms, the cryptography module available in Python was a more suitable choice. For this reason, a Flask server was created with the implementation of different algorithms that support the "did:key"

method. Flask server also imitates the actions of the delegator and utilizes the OPA for access evaluation.

4.5 Open Policy Agent

The OPA provides a convenient way to define and evaluate policies. This falls in line with the access evaluation of the defined protocol. In this implementation, OPA is implemented as a standalone server, with the help of Docker. It accepts POST requests and evaluates the access using the payload received. OPA provides policy-defining language that supports both policies and policy groups.

```
package supervisor-credential

default allow = false

allow if {

    # check if it is a university student
    input.credentialSubject.attributes.isUniversityStudent == true

    input.credentialSubject.attributes.isFromColomboUniversity == false

    # check if student id is present
    input.credentialSubject.attributes.studentId

    # check if university is present
    input.credentialSubject.attributes.university
}

allow if {

    # check if it is a university student
    input.credentialSubject.attributes.isUniversityStudent == true

    # check if student id is present
    input.credentialSubject.attributes.isFromColomboUniversity == true
}
```

Listing 8: OPA Example Policy Group

The listing 8 shows an example policy group defined for the 'supervisor-credential'. There are 2 policies defined for the credential. There are conditions defined in each policy. If either of the policies is satisfied by the ADC retrieved, it returns true. OPA allows for

defining fine-grained access control for the credential. Also, by changing a condition in one of the policies or adding a new policy to the policy group, access rules for the credential can be changed.

4.6 Use Case 01: Student Supervisor Use Case

There are three different implementations of the use case: both available methods and the defined protocol. The objective here is to measure the performance difference between these methods and measure the performance of a web based implementation of the protocol. "did:web" method is used to generate DIDs of the participating roles. DID documents of the each role is hosted in the Amazon S3 Bucket. Another important reason for selecting "did:web" method is, it allows addition of another DID as the controller while methods like "did:key" does not allow this. In addition to DID document update, "did:web" method use the HTTPS requests for DID document resolving thus making it a web based DID method. And "did:web" method conveniently supports both attenuated delegation and the defined protocol as well.

The DID manager used in this implementation is Veramo. Veramo provides all functionalities related to the "did:web" method. The Proof format used here is "jwt". The reason for selecting JWT as the proof format is that it is one of the most widely used proof formats. Figure 14 shows the operations done by each component of the use case 1 implementation.

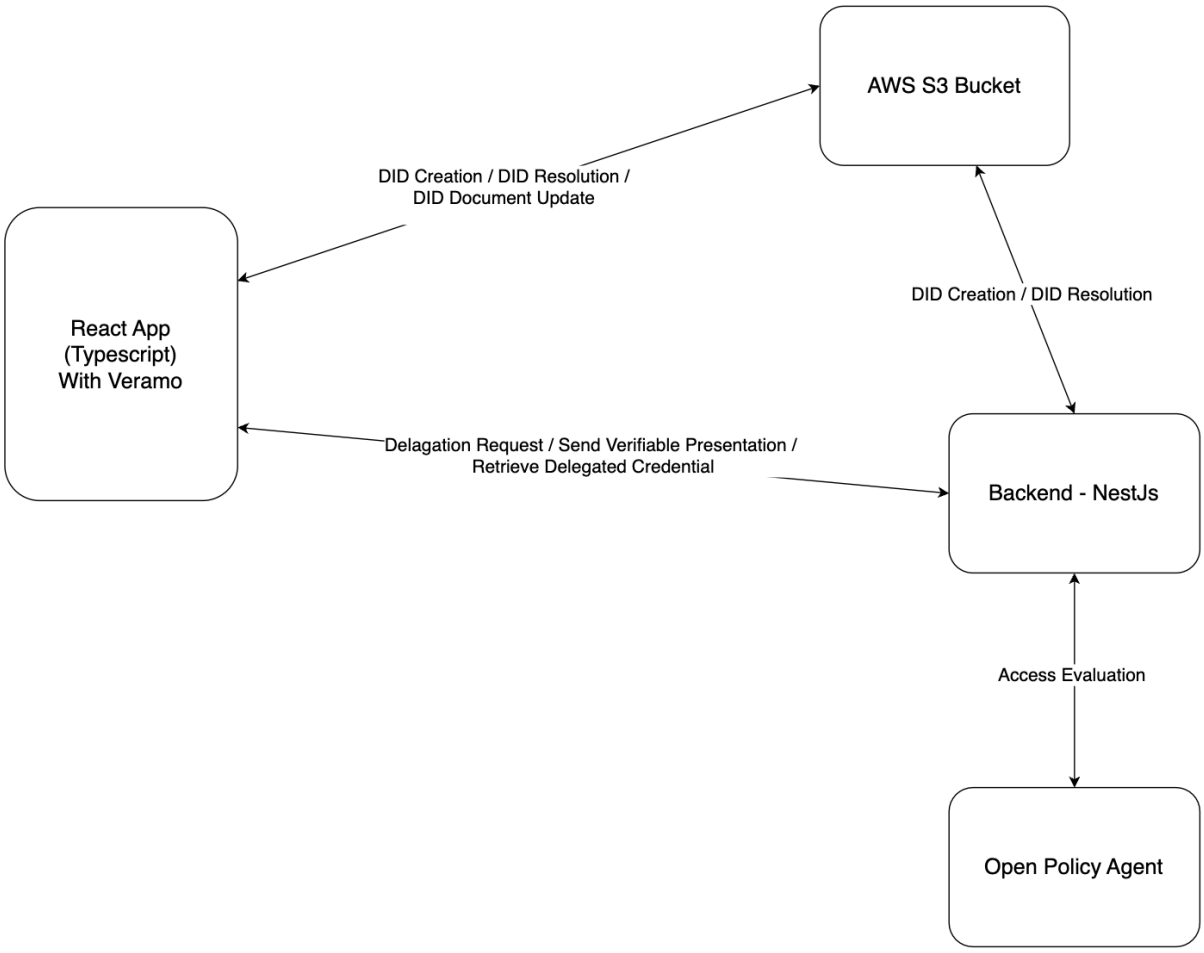


Figure 14: Use Case 1 Implementation

4.7 Use Case 02: Employee Use Case

Only the protocol is implemented in this use case. The implementation follows the "did:ethr" method using "jwt" as the proof format. For the use of "did:ethr", RPC calls for the Ethereum network should be made. This is provided by the Infura project with a daily credit limit. The "mainnet" was used for RPC. Veramo is used as the did manager, as it supports "did:ethr" as a plugin. The rest of the implementation follows the same procedure as use case 1, defined in Section 4.6, except that this use case does not implement the available 2 methods. Figure 15 shows the overview of the use case 2 implementation.

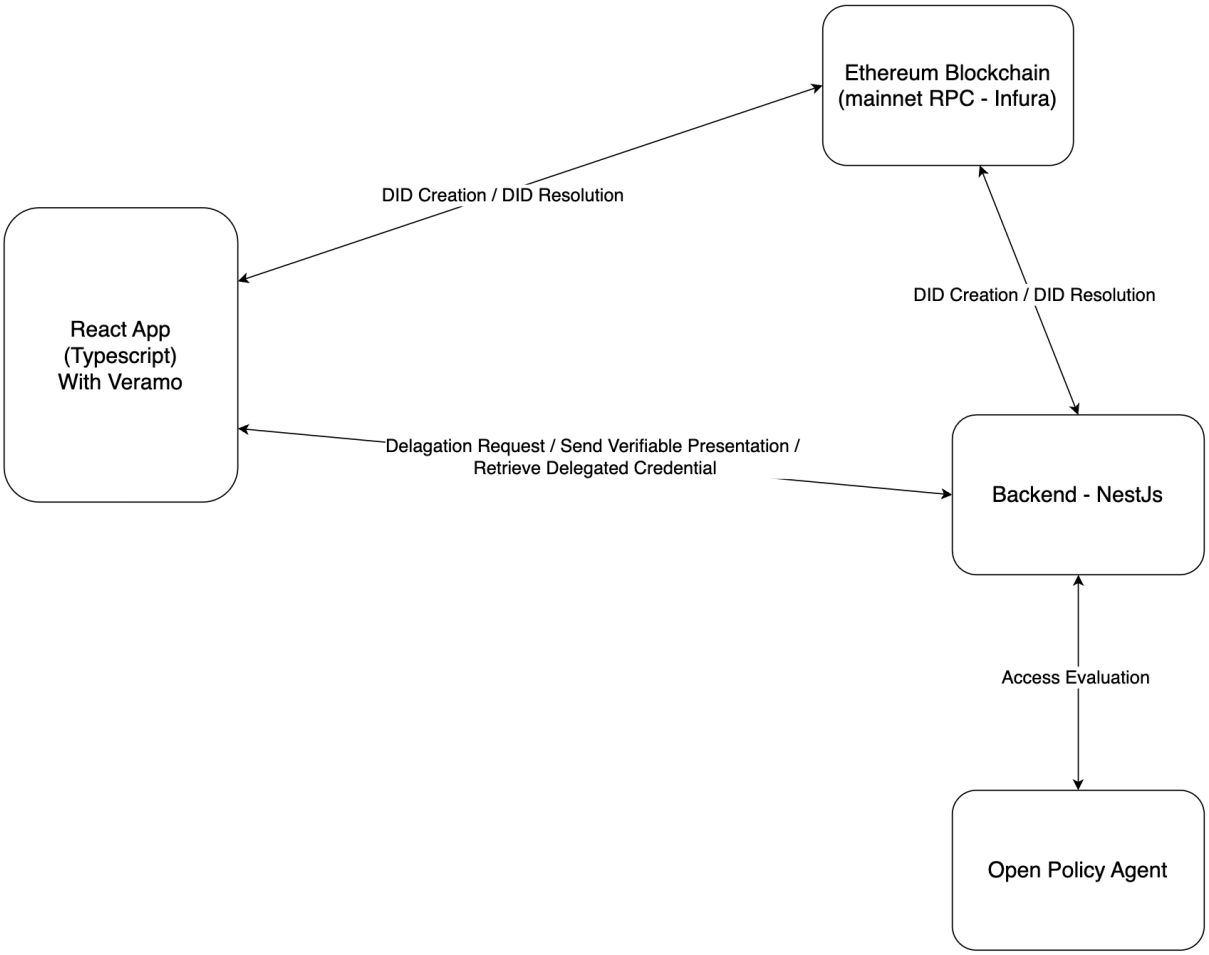


Figure 15: Use Case 2 Implementation

4.8 Algorithmic Performance Implementation

For measuring algorithmic performance, signing and verification of VCs and VPs have to be done manually due to the lack of libraries. Signing algorithms were implemented in the Python Cryptography library. The Flask framework was used to implement the backend application with the Cryptography library. In addition to the time taken, the memory & CPU utilization, and the size of the ADC were also recorded. The "did:key" method was used, since it does not have a DID document resolution mechanism that affects the readings. Another reason for choosing the "did:key" method is that it supports multiple key types. But some of the key types, like BLS12-381, are still experimental and cannot be employed for a fair evaluation. Figure 16 shows the high-level overview of the implementation.

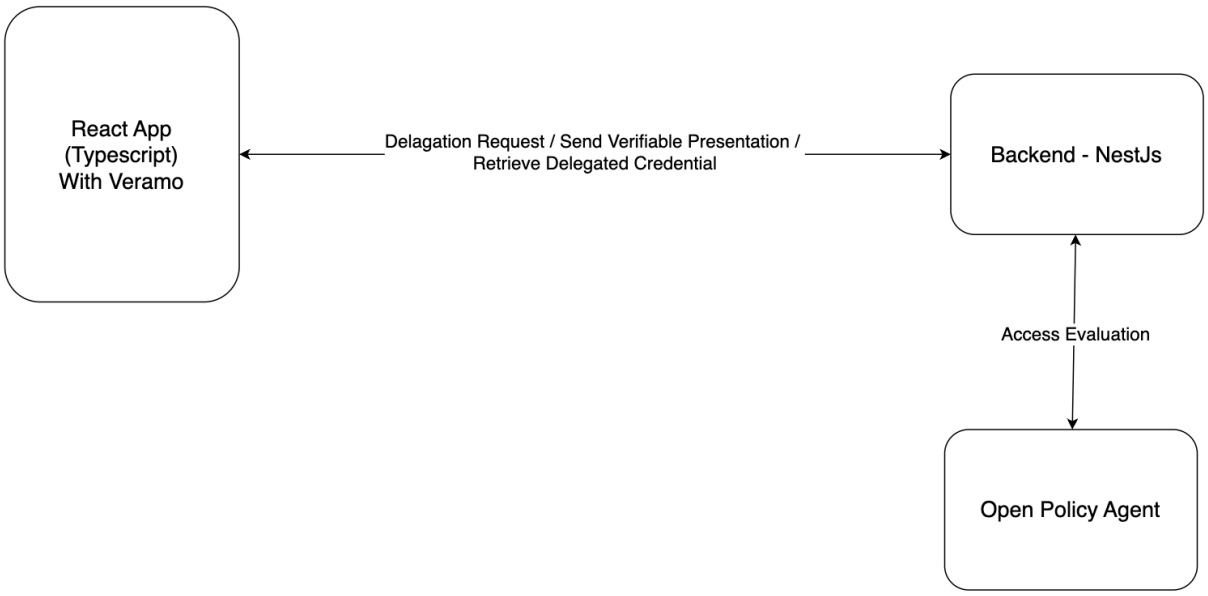


Figure 16: Algorithm Performance Implementation

Table 2 shows the breakdown of libraries and frameworks used for each implementation. These technologies were chosen by a thorough survey of available literature and testing libraries specified in them. These implementations were executed on a MacBook Air M2. This prototype only provides the minimum requirements for the protocol to work, and the real-world requirements may differ; and corresponding implementations should be adopted.

Implementation	**Front-End**	**Backend**
Student Supervisor Use Case	React with Typescript, Veramo, Axios, and AWS SDK	NestJs with Typescript, Veramo, Axios, and OPA
Employee Use Case	React with Typescript, Veramo, and Axios	NestJs with Typescript, Veramo, Axios, OPA, and Infura project 'mainnet' RPC network for Ehtereum
Algorithm Implementation	React with Typescript, and Axios	Flask with Cryptography library, and OPA

Table 2: Implementation Technologies

5 Results

For a fair evaluation, each use case was executed 100 iterations, and focused metrics were recorded. The catching functionalities of the used frameworks were disabled. Selected DID methods utilize web technologies, so the latencies in receiving responses affect the readings. To remove these outliers, the dataset collected was cleaned by removing all data points outside the interquartile range, and then the datasets were balanced by adjusting their size to the minimum of them.

5.1 Results From Use Cases

The reading from the use case implementation is summarized in Table 3. The average of each step was taken and rounded to 2 decimal points.

Use Case	**Average Delegation Time (ms)**	**Average Verification Time (ms)**	**Average Total Time (ms)**
Supervisor Student Attenuated	114.94	114.78	229.72
Supervisor Student DID Document	227.61	161.62	389.22
Supervisor Student Protocol	154.05	313.28	467.33
Employee Protocol	557.00	4254.92	4811.92

Table 3: Use Case Readings

The expected comparison for the student supervisor use case is between the available methods and the protocol. It is summarised in Figure 17. Each method varies in performance in delegation and verification. But for the total time taken, there is a clear distinction. The protocol performance on different DID implementations is shown in Figure 18. There is a significant difference observed in these two implementations. The reasons for these observations will be discussed in Section 6.

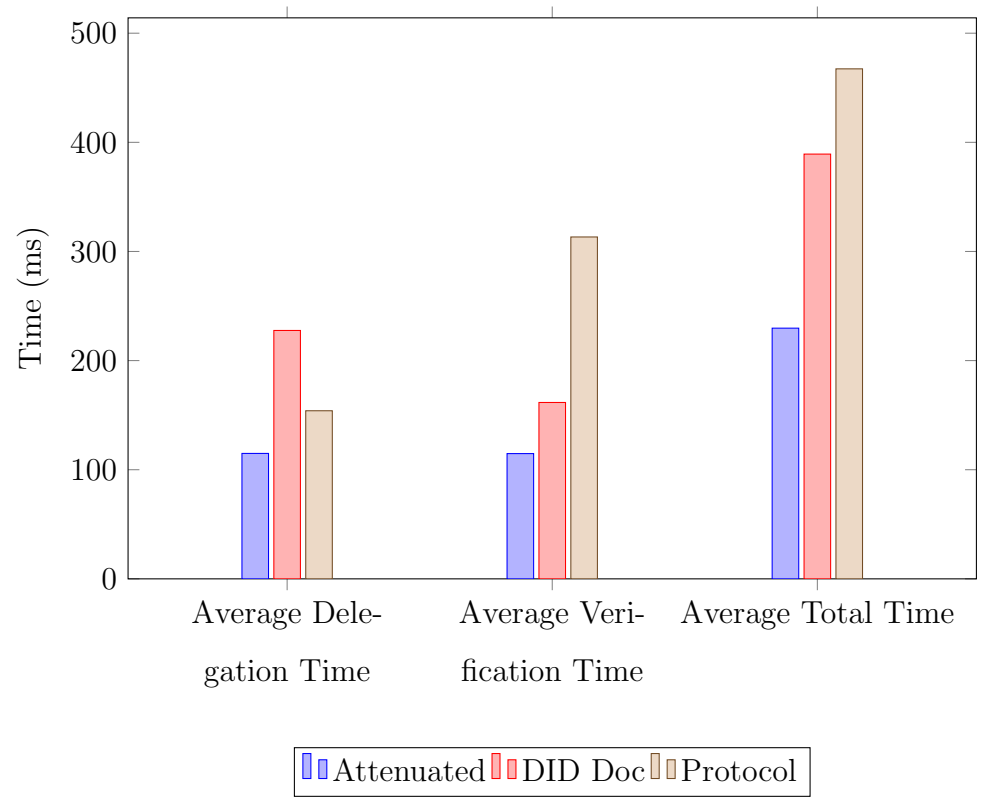


Figure 17: Delegation Method Comparison

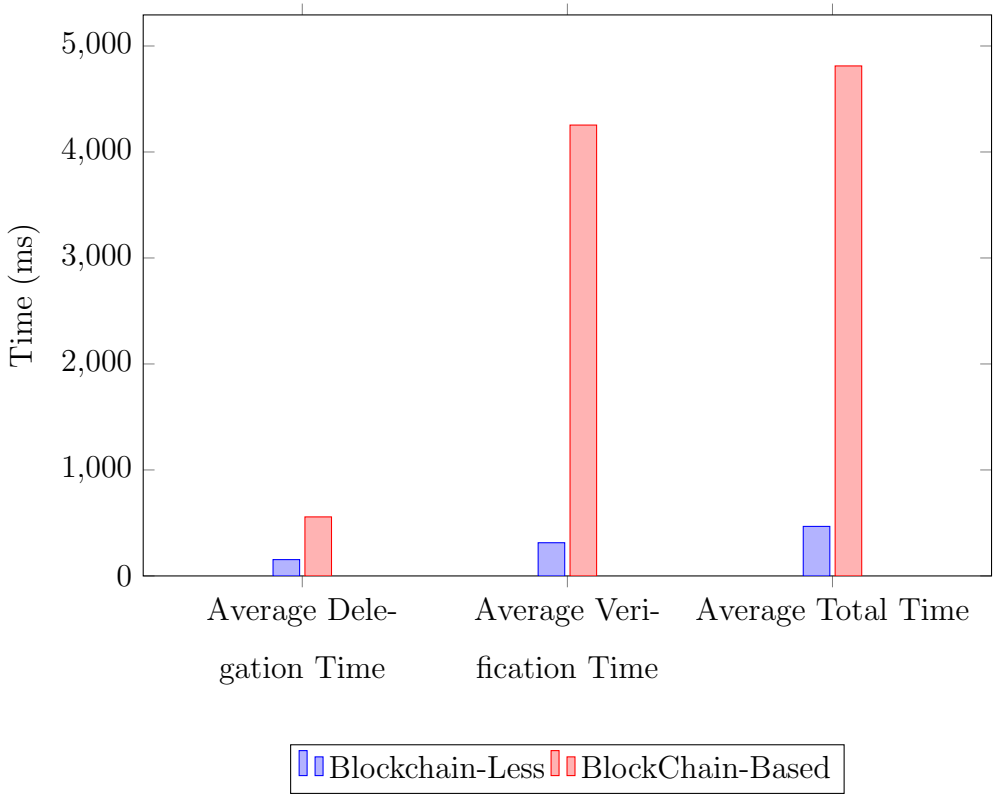


Figure 18: DID Method Comparison

In addition to these measurements, since the employee use case is implemented using

the Ethereum blockchain RPC, there is a cost for these RPC calls. This cost is associated with DID creation and DID document resolution. Cost incurred for the evaluation process is shown in Figure 19, which was taken from the Infure project dashboard.

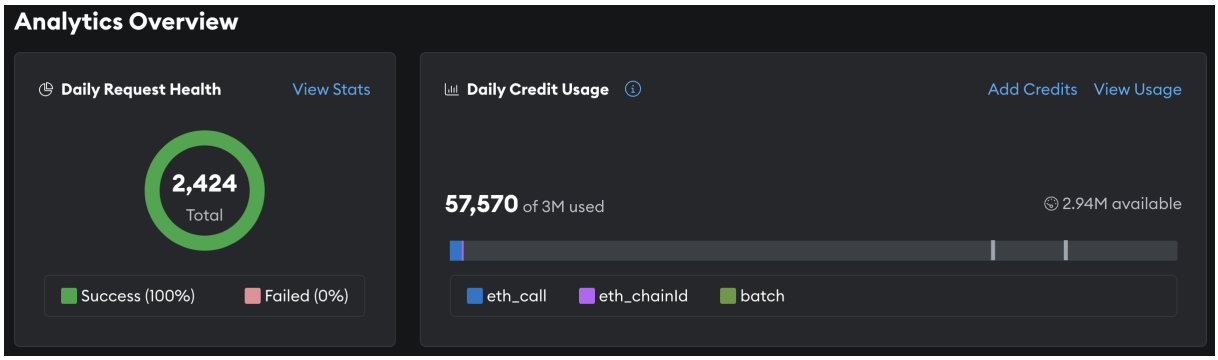


Figure 19: Cost Incurred for RPC Calls

5.2 Results From Key Type Implementations

Reading from the key type implementation includes the size of the ADC, containing the same claims, time taken, and memory & CPU usage. Since there is no DID document resolution, the CPU utilized by this process is accurate due to a lack of I/O interruptions. And the memory usage includes the memory used by that specific process (code, stack, and heap). The readings are summarized in Table 4

Reading	**Ed25119**	**Secp256k1**	**P-256**
ADC Size (in bytes)	719	823	808
Average Delegation Time (seconds)	0.43482	1.09565	0.47594
Average Delegation CPU Usage (seconds)	0.00046	0.00112	0.00050
Peak Delegation Memory Usage (in KB)	384	32	48
Average Verification Time (seconds)	0.26702	0.52658	0.26782
Average Verification CPU Usage (seconds)	0.00030	0.00056	0.00030
Peak Verification Memory Usage (in KB)	384	16	16
Average Retrieval Time (seconds)	7.26495	7.68064	7.97954
Average Retrieval CPU Usage (seconds)	0.00290	0.00314	0.00312
Peak Delegation Memory Usage (in KB)	1200	304	496

Table 4: Key Type Readings

Readings in Table 4 are compared against each algorithm in the figures 20, 21, and

22. Even though these readings do not explain the performance of the protocol, they are vital when building an implementation. Their importance will be discussed in Section 6.

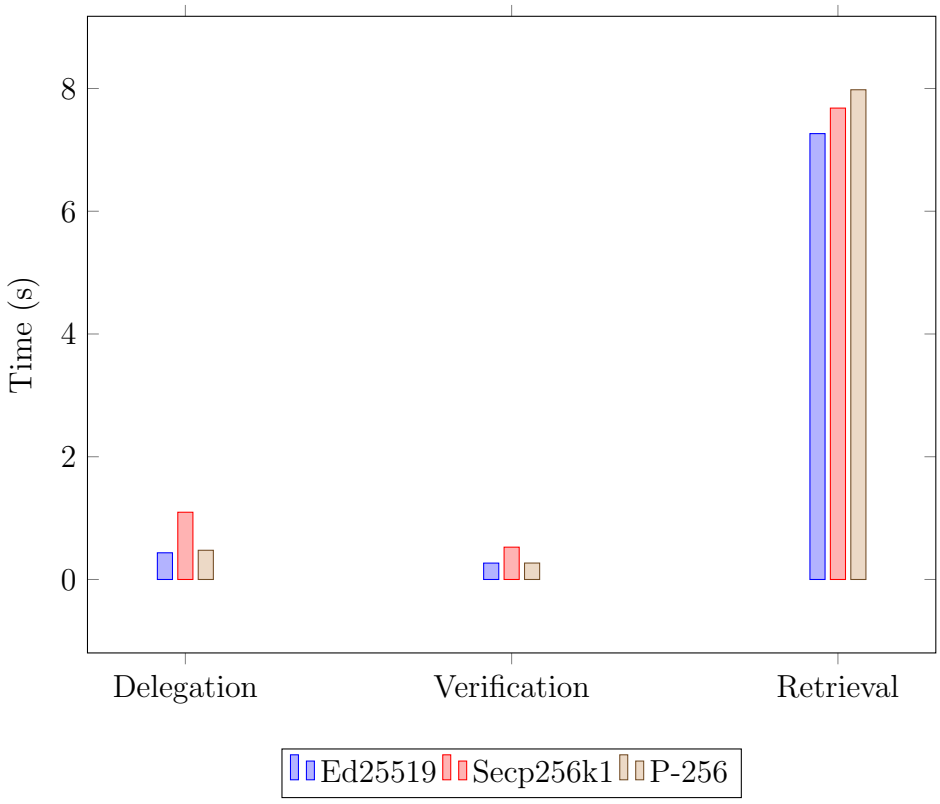


Figure 20: Time Taken for Delegation, Verification, and Retrieval

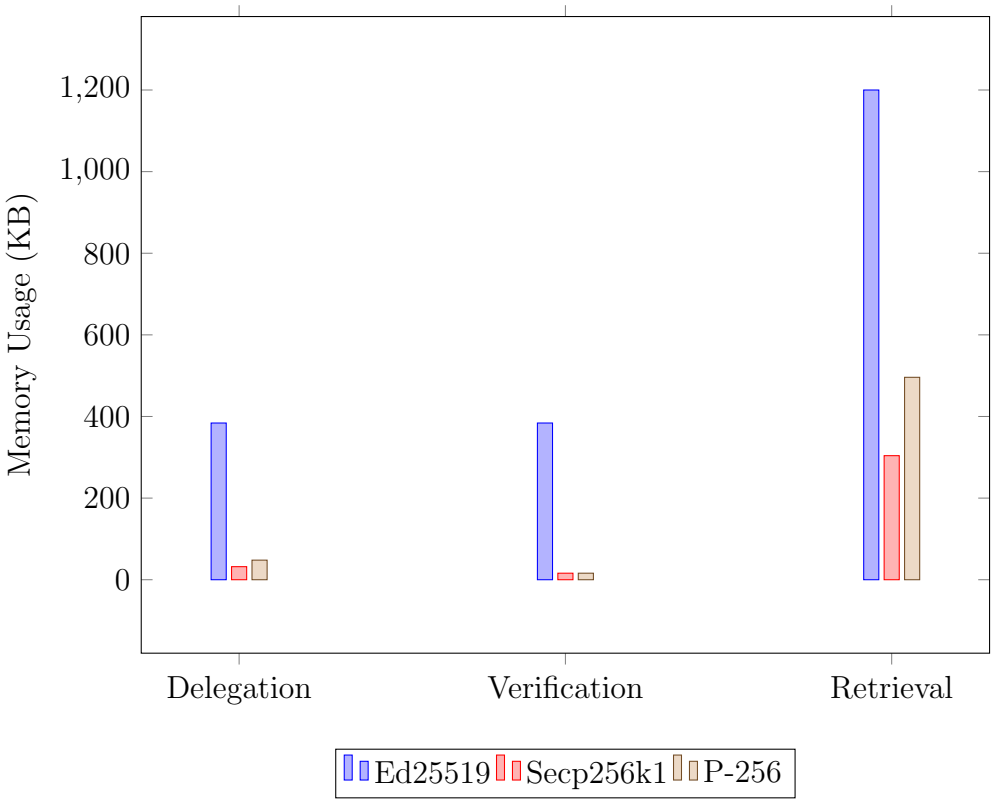


Figure 21: Memory Usage for Delegation, Verification, and Retrieval

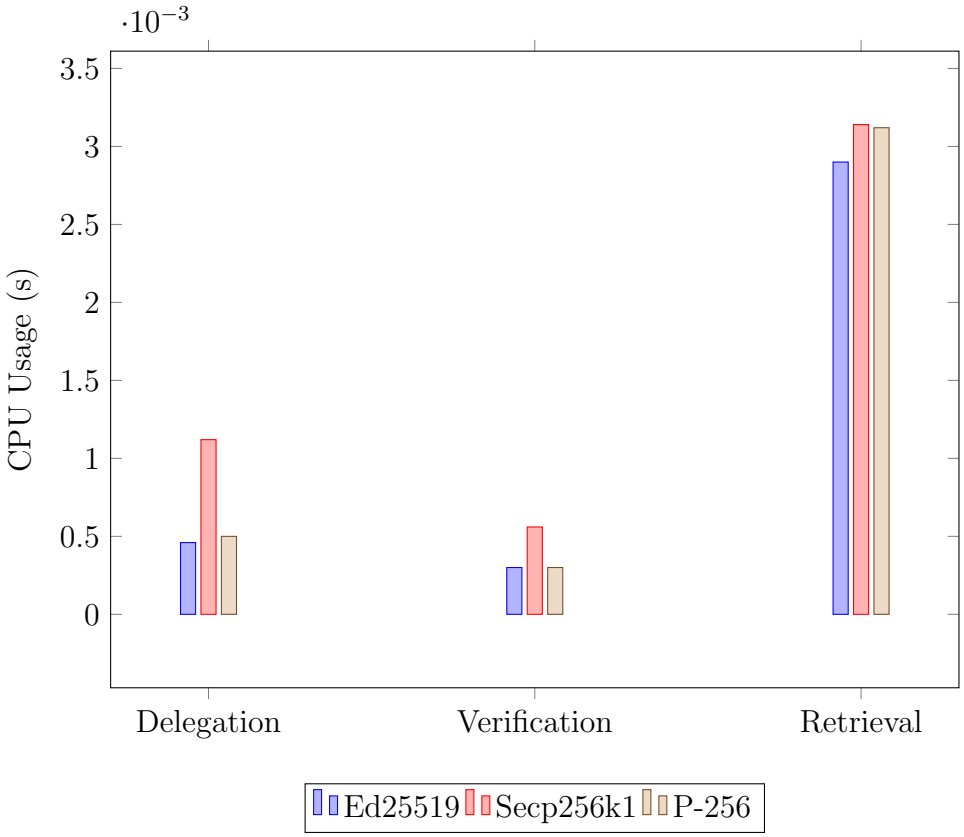


Figure 22: CPU Usage for Delegation, Verification, and Retrieval

A qualitative analysis of the protocol was conducted to check its adherence with the privacy requirements specified in Section 1.8.1. Though the purpose of this protocol is to address the privacy issues mentioned in Section 1.3, there are still some privacy concerns that should be addressed in terms of implementation. They will be discussed in Section 6.

6 Discussion

To answer the first 2 research questions, an in-depth analysis of the chosen DID methods out of over 150 DID methods was carried out. Since the nature of the implementation of each DID method, not all of them are suitable for the implementation of a dynamic protocol. It is noteworthy to mention that even though there are over 150 methods [14], some of the DID methods are incomplete, and the cost for registering records significantly differs from one another (as shown in Figure 19). After surveying specifications of the majority of DID methods, it was revealed that some DID methods do not support the update of the DID document, and some do not support recovery of the DID. Since methods like updating the DID document are supported by some DID methods, it should not

be accounted for in the protocol development. Most of the blockchain-based implementations that use blockchains like Ethereum and Bitcoin have costs related to them that will affect solutions that use them. And the time taken for operations related to the DID method significantly differs among these methods. For example, the BTC Ordinals DID method's specification mentions that it might take up to 10 minutes for some operations because the Bitcoin blockchain has grown considerably. The DID methods "did:web", "did:ethr", and "did:key" were chosen for the implementation as they conform to the specifications mentioned by W3C as per the majority of DID methods and they have already implemented libraries that can be use.

In addition to these specifications of DID methods, a survey on available implementations related to DIDs and VCs was conducted on available literature and on the web as well. According to [12], Veramo, which is a result of uPort, Aries Cloud Agent (ACA), Hyperledger Aries, and DIDKit, are implementation-ready libraries that are free to use. Even though [12] described them as implementation-ready, upon testing them for the evaluation implementation, it was revealed that they still need to be evolved to be production-ready. For example, ACA received continuous breaking updates during the testing period, including a change of core libraries that it used. Though this implementation includes Veramo, some functionalities that will be needed for future work (described in section 9) are not yet implemented or already implemented but susceptible to changes according to their documentation. Choosing suitable technologies for the implementation was itself a noteworthy challenge, as a continuous literature survey revealed that many possible techniques and technologies can be used to achieve certain requirements, but the implementations are still novice and will continue to change until they become stable.

Taking all previous knowledge into consideration, the protocol defined in Section 3 was developed. The protocol itself does not violate the SSI principle. The VC to be delegated will not be transferred to the delegatee, and any claims included in that VC will not be revealed to any party other than the verifier. This protocol answers the first research question, that it is possible to develop a protocol that does not violate the SSI principle, which enables the access control delegation. This protocol is supported by several SSI related technologies, but not their specific implementations. So there is no barrier to implementing this protocol, and it may even be customized to meet special requirements. The dynamic nature of the access control delegation is done by ABAC, which is one of the access control mechanisms that is being used among access control solutions. ABAC aligns with VCs, as VCs contain claims which are similar to attributes. This makes

ABAC work in SSI solutions. In addition to ABAC, the credential itself is dependent on DID related technologies. This is because the issuer and the subject of a VC are always identified by their DID. In addition to these technologies, the DIDComm protocol can be used for more secure communication. These methodologies allow the protocol to work. Thus, they answer the second research question of methodologies that enable dynamic access control delegation for sharing verifiable credentials.

The performance evaluation of this protocol will be discussed in Section 6.1, and the privacy evaluation will be discussed in Section 6.2.

6.1 Performance Evaluation Discussion

The time taken comparison between available methods and the protocol is shown in Figure 17. Attenuated delegation has the least impact on time, and the designed protocol performs the worst. The average time taken for delegation is highest for the adding delegatee as the controller method. It is because of the update of the DID document. This method requires the DID document to be updated wherever it is hosted. Since the "did:web" method is used, the difference in time taken is not vast. But if it were to use a DID method that employs a blockchain, this difference would be noteworthy. Unlike the other 2 methods, this delegation will not affect only a single credential, but it delegates the DID of the delegator along with any VC of the delegator. The reason for the observation of the time difference between attenuated delegation and the protocol is the issuance of ADC. This issuance requires verifying the delegatee's VP and creating claims for the access of the VC.

The average verification time is greatest for the protocol. This is because after verifying the VP presented by the delegate, the verifier should retrieve the DC using the ADC. The reason for the difference in time taken for the other 2 methods is that adding the delegatee as the controller updates the DID document, and for the verification, the new public key (of the controller) should be resolved. There is a clear difference between methods in the total time taken. Though the protocol performs worse, it preserves the SSI principle, unlike the other 2 methods. It is also noteworthy that this difference will change according to the DID method used, but the comparison will be the same.

The comparison of the time taken for the protocol in different DID implementations is shown in Figure 18. The target here is a blockchain-less implementation and a blockchain-based implementation. The bar graph shows a clear difference between the two methods.

This is due to the use of RPC calls on a public blockchain over simple HTTP requests. A blockchain-based implementation that uses the Bitcoin blockchain will show an even greater time difference. However, most of the solutions available prefer methods like "did:indy" and "did:ethr" because the blockchain acts as a verifiable data registry. A production-ready implementation should consider the implications of using the aforementioned DID methods along with the cost that is incurred with the blockchain.

The results from the key type implementations are summarized in Table 4. There is no significant change in the size of the ADC that can affect an implementation. The focus should be on the time taken, memory, and CPU usage. All of them show low numbers due to the usage of "did:key" which does not require a DID document resolution. Ed25519 is the key type that is widely used in the majority of the implementations. It performs best out of the 3 key types on time taken and CPU usage. But it has considerable memory usage in comparison. In terms of time taken and CPU usage, Secp256k1 performs worse. But it is not by a considerable amount. Among all 3 key types, the retrieval step performs worse. This is due to the access evaluations (using OPA server), which take place during the retrieval step.

The reason for focusing on these steps with the key type is that when implementing the protocol, there are considerations that should be taken into account. This reading is from a minimalistic and automated implementation. The solutions that are needed in the real world may differ. And the system (mobile device or a server) on each step can also differ. The load on the system should be taken into account along with the DID methods used. Key types that are still experimental have not been tested here.

6.2 Privacy Evaluation Discussion

Evaluation of the protocol under the privacy requirements specified in Section 1.8.1 are summarized below.

6.2.1 Multi-Show Unlinkability

"several credential presentations derived from the same original credential and transmitted over several sessions cannot be linked by verifiers"

Satisfaction of this requirement depends on the implementation of the protocol, specifically the implementation of the issuance of the ADC. The weakness here is the "credentialId" key, if this is a unique static ID that can be identified across several verifies

over several sessions, there is the risk of likability. By implementing dynamic identifiers that can be identified at the delegator's service endpoint, this can be prevented. Another weakness is the DID of the delegatee present in the ADC. But due to the pseudonymity property of DID, this can be prevented by using different identifiers for each session. Still, there is a risk of linking different identifiers with each other through the careless use of them.

There is no barrier for the use of privacy-preserving schemas like ZKP in the ADC. This applies to the "attributes" key, which is only needed for the delegator. Therefore, if needs, it can be encrypted and included in the ADC. But the values under "service" key should be visible to the verifier, as the "serviceEndpoint" is used to retrieve the DC. This can also be a source of likability, but it can be prevented by anonymity-preserving protocols like Tor. The protocol adheres to the multi-show linkability requirement but heavily depends on the implementation.

6.2.2 Issue-Show Unlinkability

"any information collected at the time of credential issuance cannot be used subsequently to establish a link between the credential presentation and the original credential"

At the issuance of the ADC, VP is presented by the delegatee showing their identity to the delegator. ADC is issued after verifying these claims in the VP. If ADC contains claims collected from the VP, it will act as a link to the original credentials present in the VP. This can also be prevented by ZKP and other privacy-preserving schemes like AnonCreds or BBS+. There is also the issuance timestamp, which reveals the time of the issuance.

If the same ADC is used across several sessions, it can reveal metadata collected at the issuance, depending on how they are used. This metadata may be tightly coupled with the delegatee's DID, which may be used elsewhere, which can be a point for linking. Issue-show unlinkability is also partially satisfied by the protocol, but it depends on the implementation.

6.2.3 Non-Correlation

"non-correlation is the inability of any actor, including curious actors on the infrastructure, for example, or providing a service like DNS servers, to learn about the activity of the holder by linking their different identifiers together, or by linking an activity to the

specific holder"

Use of static values in the ADC will be the source for correlation. For example, a static "serviceEndpoint" can reveal the actors trying to access the same credential. There is the option of using the DIDComm protocol, allowing an extra layer of security and privacy for the communication between actors. Correlation is one of the main privacy-related problems that is present across many SSI related technologies. Since the protocol utilizes these available technologies, any technique that is being applied can be incorporated into this solution.

As opposed to FIM solutions, correlation has been made extremely difficult in SSI solutions. This protocol also follows the same principles used by other SSI solutions. Therefore, the protocol adheres to the non-correlation requirement. The implementation and the knowledge of the actors on using credentials using best practices play a key role in satisfying this requirement.

6.2.4 Ability to Use Selective Disclosure Techniques

There is no barrier for using selective disclosure techniques or ZKPs in this protocol implementation. The ADC complements them, but "service" should be visible to the verifier. It is a weakness that can reveal the relationship between the delegator and the delegatee, as the DC is retrieved from this service endpoint. But it is the expected behaviour, so "service" should be visible to the verifier. There is no way around where masking of the "service" is an option. Except for the "service" attribute, all other values in the ADC support selective disclosure.

6.2.5 Transparency

"transparency is fundamental for privacy and data protection, since it guarantees that the data subject is aware of how their data is being processed and where it is stored, etc."

One of the main goals of this protocol is to make the delegator known when the delegator's credentials are being used after delegation. The protocol achieves this. In addition to the transparency of the credential used, the delegator is involved in the use of the DC. The delegator also possesses the ability to intervene and revoke the access anytime after the delegation. This can be a policy change or a revocation list.

The protocol strongly supports transparency and is designed for it. And it transferred the control of the DC credential back to the subject of the credential, unlike previously

used methods.

6.2.6 Non-Traceability

"non-traceability means that a curious actor is not able to trace the activities of a certain user back to them"

The protocol does not eliminate the possibility of tracing by a curious actor. A curious verifier can track the delegations between the delegator and the delegatee using the "service" present in the ADC. This protocol introduces a source of tracing for the verifier. And also, the verifier can identify the actions of delegates by tracing verifiers that access the "serviceEndpoint". The intention behind the protocol design is to allow the delegator to know when the DC is being used. It can be exploited by the dishonest delegator.

Tracing done by the delegator cannot be prevented, but the tracing done by the verifier can still be prevented using privacy-preserving techniques implemented at the "serviceEndpoint". The protocol does not strictly adhere to the non-traceability requirement.

7 Conclusion

SSI solutions and related technologies are still in their early stages of development and will continue to evolve during the next few years. Currently implemented solutions are also undergoing significant changes and will continue to do so until present problems are solved. This project was aimed at solving the problem of delegation that is present in the currently employed methods. The protocol specified in Section 3 solves the problems of delegation, but it is not a complete solution.

The protocol performs worse than the already available methods due to the involvement of the delegator in the credential presentation flow. Some use cases prefer the delegation by adding the delegatee as the controller, because it is not a repetitive process for each credential. But the protocol does achieve the intended privacy concerns and prevents the unintended use of the DC by the delegatee, providing transparency to the delegator. And it adheres to the majority of the privacy by design requirements specified by [13]. The implementations of the protocol will change according to the real-world requirements. And future changes to employed methods may expand the room for development of the protocol.

8 Limitations

As discussed in Sections 6 and 7, the protocol satisfies the objective of this project. But there are limitations, especially regarding the "service" attribute in the ADC. But according to the design of the protocol, these limitations cannot be overcome easily. It may need to change the approach significantly. The protocol design itself had constraints when developing due to interoperability between available SSI standards. Going further away from these standards will provide a hindrance to implementation, same way it has been for the implementation of universal DID resolver.

Due to time constraints and lack of availability of implementation, the protocol has not been evaluated under a majority of the DID methods available. And also, there are some key types that are under experimentation that may affect the implementation of the protocol differently. This protocol is an adaptation of the currently available SSI related technologies to solve the privacy-related problem of the delegation of VCs.

9 Future Work

Due to a lack of stable libraries that provide SSI implementation, evaluation of the protocol was limited. This protocol can be further evaluated using other DID methods that are being used. In addition to DID methods, there are other proof formats, some lack implementation, and some are still under development, that can be employed in the protocol. A thorough evaluation of the protocol should be done by implementing it. Blind signatures like BBS+ provide new possibilities for the designed protocol and may allow for further privacy protection. One of the unresolved problems in SSI is, use of organization credentials by employees with an outside entity. This protocol may provide a foundation for a solution to the problem.

The implementation done in this project only follows an automated issuance of ADC, and intervention at retrieval. This should be further explored to employ both automated and active participation from the delegator. There is no barrier to said implementation. But it should be explored for possible privacy and security vulnerabilities, together with performance implications. This protocol is viable for the current ecosystem of SSI technologies. But since the world of SSI continues to evolve at a rapid pace, a revisitation of the protocol may be needed. Continuous adaptation to SSI related technologies will ensure its viability.

References

[1] N. Naik and P. Jenkins, "Self-sovereign identity specifications: Govern your identity through your digital wallet using blockchain technology," in *2020 8th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, IEEE, 2020, pp. 90–95.

[2] Y. Ding, J. Yu, S. Li, H. Sato, and M. G. Machizawa, "Data aggregation management with self-sovereign identity in decentralized networks," *IEEE Transactions on Network and Service Management*, 2024.

[3] M. Sporny, I. Herman, D. Chadwick, and D. Longely, *Verifiable credentials data model v2.0*, 2025. [Online]. Available: `https://www.w3.org/TR/vc-data-model-2.0/`.

[4] M. Sporny, D. Longely, M. Sebadello, D. Reed, O. Steele, and C. Allen, *Decentralized identifiers (dids) v1.0*, Jul. 2022. [Online]. Available: `https://www.w3.org/TR/did-1.0/`.

[5] M. Niemiec and W. Kolucka-Szypula, "Federated identity in real-life applications," in *2015 European Conference on Networks and Communications (EuCNC)*, IEEE, 2015, pp. 492–496.

[6] S. Lim, M.-H. Rhie, D. Hwang, and K.-H. Kim, "A subject-centric credential management method based on the verifiable credentials," in *2021 International Conference on Information Networking (ICOIN)*, IEEE, 2021, pp. 508–510.

[7] R. De Prisco, S. Shevchenko, and P. Faruolo, "Enhancing openid connect for verifiable credentials with didcomm," 2024.

[8] J. Jensen, "Federated identity management challenges," in *2012 Seventh International Conference on Availability, Reliability and Security*, IEEE, 2012, pp. 230–235.

[9] W. Li and C. J. Mitchell, "User access privacy in oauth 2.0 and openid connect," in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, IEEE, 2020, pp. 664–6732.

[10] G. Kondova and J. Erbguth, "Self-sovereign identity on public blockchains and the gdpr," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 2020, pp. 342–345.

[11] V. R. Doncel, "Web technologies for decentralised identity,"

[12] C. Mazzocca, A. Acar, S. Uluagac, R. Montanari, P. Bellavista, and M. Conti, "A survey on decentralized identifiers and verifiable credentials," *arXiv preprint arXiv:2402.02455*, 2024.

[13] M. Naghmouchi and M. Laurent, "Privacy by design for self-sovereign identity systems: An in-depth component analysis completed by a design assistance dashboard," *arXiv preprint arXiv:2502.02520*, 2025.

[14] S. Bistarelli, F. Micheli, and F. Santini, "A survey on decentralized identifier methods for self sovereign identity.," in *ITASEC*, 2023.

[15] R. Hörbe and W. Hötzendorfer, "Privacy by design in federated identity management," in *2015 IEEE Security and Privacy Workshops*, IEEE, 2015, pp. 167–174.

[16] E. Ghazizadeh, M. Zamani, A. Pashang, *et al.*, "A survey on security issues of federated identity in the cloud computing," in *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, IEEE, 2012, pp. 532–565.

[17] R. Nokhbeh Zaeem, K. C. Chang, T.-C. Huang, *et al.*, "Blockchain-based self-sovereign identity: Survey, requirements, use-cases, and comparative study," in *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, 2021, pp. 128–135.

[18] N. Naik and P. Jenkins, "Uport open-source identity management system: An assessment of self-sovereign identity and user-centric data platform built on blockchain," in *2020 IEEE International Symposium on Systems Engineering (ISSE)*, IEEE, 2020, pp. 1–7.

[19] P. J. Windley, "Sovrin: An identity metasystem for self-sovereign identity," *Frontiers in Blockchain*, vol. 4, p. 626 726, 2021.

[20] D. Maldonado-Ruiz, J. Torres, and N. El Madhoun, "3bi-ecc: A decentralized identity framework based on blockchain technology and elliptic curve cryptography," in *2020 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, IEEE, 2020, pp. 45–46.

[21] G. Alpár, F. Van Den Broek, B. Hampiholi, B. Jacobs, W. Lueks, and S. Ringers, "Irma: Practical, decentralized and privacy-friendly identity management using smartphones," in *10th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2017)*, 2017, pp. 1–2.

[22] P. Dunphy and F. A. Petitcolas, "A first look at identity management schemes on the blockchain," *IEEE security & privacy*, vol. 16, no. 4, pp. 20–29, 2018.

[23] J. J. Jeong, R. Doss, L.-x. Yang, *et al.*, "Addressing the privacy by use challenges in verifiable credential based digital wallets," 2024.

[24] R. Laborde, A. Oglaza, S. Wazan, *et al.*, "A user-centric identity management framework based on the w3c verifiable credentials and the fido universal authentication framework," in *2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC)*, IEEE, 2020, pp. 1–8.

[25] A. Flamini, G. Sciarretta, M. Scuro, A. Sharif, A. Tomasi, and S. Ranise, "On cryptographic mechanisms for the selective disclosure of verifiable credentials," *Journal of Information Security and Applications*, vol. 83, p. 103 789, 2024.

[26] Š. B. Ramić, E. Cogo, I. Prazina, *et al.*, "Selective disclosure in digital credentials: A review," *ICT Express*, 2024.

[27] R. Mukta, J. Martens, H.-y. Paik, Q. Lu, and S. S. Kanhere, "Blockchain-based verifiable credential sharing with selective disclosure," in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, IEEE, 2020, pp. 959–966.

[28] T. Wang, W. Fang, and W. Zhang, "Access control for iot-based big data: A state-of-the-art review," in *Proceedings of the 2024 6th International Conference on Big Data Engineering*, 2024, pp. 81–87.

[29] P. W. Fong, "Relationship-based access control: Protection model and policy language," in *Proceedings of the first ACM conference on Data and application security and privacy*, 2011, pp. 191–202.

[30] H.-b. Shen and F. Hong, "An attribute-based access control model for web services," in *2006 Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'06)*, IEEE, 2006, pp. 74–79.

[31] A. Sissodiya, U. Bodin, and O. Schelén, "Objective-and utility-based negotiation for access control," in *11th International Conference on Information Systems Security and Privacy, Porto, Portugal, February 20-22, 2025*, Science and Technology Publications, Lda, vol. 2, 2025, pp. 493–501.

A List of DID Methods

1. 3
2. abt
3. aergo
4. ala
5. amo
6. antelope
7. art
8. asset
9. bba
10. bee
11. bid
12. bluetoqueagent
13. bluetoquedeed
14. bluetoquenfe
15. bluetoqueproc
16. bnb
17. bryk
18. btcr
19. ccf
20. ccp
21. celo
22. cheqd

23. com
24. corda
25. cosmos
26. cot
27. cr
28. did
29. dns
30. dock
31. dom
32. dsrv
33. dual
34. dxd
35. dyne
36. echo
37. elastos
38. elem
39. emtrust
40. ens
41. eosio
42. erc725
43. etho
44. ethr
45. ev
46. evan

47. everscale
48. example
49. factcom
50. fairx
51. future
52. gatc
53. gns
54. grg
55. grn
56. health
57. hedera
58. holo
59. hpass
60. hsk
61. iamx
62. ibmdc
63. icon
64. id
65. iid
66. indy
67. infra
68. io
69. ion
70. iota

71. ipid
72. is
73. iscc
74. iwt
75. jilinc
76. jnctn
77. jolo
78. jwk
79. keri
80. key
81. kilt
82. klay
83. kr
84. kscirc
85. lac
86. life
87. lit
88. meme
89. meta
90. moac
91. monid
92. morpheus
93. mydata
94. near

95. next
96. nft
97. nuggets
98. nuts
99. object
100. ockam
101. omn
102. onion
103. ont
104. op
105. orb
106. oyd
107. panacea
108. peaq
109. peer
110. pid
111. pistis
112. pkh
113. pml
114. polygon
115. polygonid
116. prism
117. psi
118. psqr

119. ptn
120. qes
121. qui
122. ray
123. real
124. rm
125. safe
126. san
127. schema
128. scid
129. self
130. selfkey
131. sideos
132. signor
133. sirius
134. snail
135. snplab
136. sol
137. sov
138. ssb
139. ssw
140. stack
141. tangle
142. tdid

143. ti
144. tls
145. trust
146. trustbloc
147. trx
148. ttm
149. twit
150. tyron
151. tys
152. tz
153. unik
154. unisot
155. uns
156. uport
157. v1
158. vaa
159. vaultie
160. vertu
161. vid
162. vivid
163. vtid
164. vvo
165. web
166. wlk

167. work

168. zk