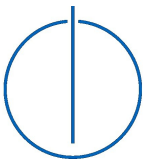
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

On the applicability of emerging identity standards to decentralized identity directories

Tristan Schwieren



DEPARTMENT OF INFORMATICS

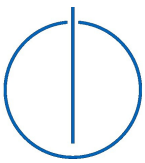
TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

On the applicability of emerging identity standards to decentralized identity directories

Über die Anwendbarkeit von entstehenden Identitätsstandards für dezentrale Identitätsverzeichnisse

Author:	Tristan Schwieren
Supervisor:	Prof. Dr. Claudia Eckert
Advisor:	Dr. rer. nat. Martin Schanzenbach
Submission Date:	15.2.2022



I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Garching bei München, 15.2.2022 Tristan Schwieren

Acknowledgments

I would like to thank my advisor Martin Schanzenbach for helping me day and night. His assistance throughout the entire process made this thesis possible. I also thank Leni Rohe, Leon Windheuser, and Valerie Csanady for reading my thesis pre-submission and giving me feedback.

Abstract

New emerging identity standards such as Decentralized Identities and Verifiable Credentials are being developed. While some people see them as a way to give internet users more privacy and freedom, others criticize them for using either centralized or environmentally harmful technology. In this bachelor thesis, we are going to explore how the new identity standards can be used in the GNU Name System and re:claimID. Both systems are not environmentally harmful or centralized while still protecting users' privacy and freedom. We show how Decentralized Identifiers can be used with the GNU Name System as the underlying Verifiable Data Registry. We then map Verifiable Credentials to re:claimID and explain some of the arisen problems. We are going to look at other existing systems implementing Decentralized Identifiers. We are going to find out that some standards are already very useful while others are still early in their development. Last, we will describe our findings on the applicability of the emerging identity standards.

Contents

Acknowledgments **iii**

Abstract **iv**

1 Introduction **1**

2 Background **3**
2.1 Decentralized Identifier . . . 3
2.1.1 DID Syntax . . . 3
2.1.2 DID Methods . . . 5
2.1.3 DID Documents . . . 6
2.1.4 Criticism . . . 7
2.2 Verifiable Credentials . . . 8
2.2.1 Verifiable Presentations . . . 9
2.3 Verifiable Data Registry . . . 11
2.4 GNUnet . . . 12
2.4.1 GNU Name System (GNS) . . . 12
2.4.2 GNUnet Identity Subsystem . . . 13
2.4.3 re:claimID . . . 13
2.5 Organisations working on new Identity Standards . . . 14

3 Related Work **17**
3.1 Bitcoin DID Method . . . 17
3.2 Web DID Method . . . 19
3.3 Hyperledger Aries and Indy . . . 19

4 Mapping Self-Sovereign Identity Standards to re:claimID **21**
4.1 Approach . . . 21
4.2 re:claimID Decentralized Identifier Method . . . 21
4.2.1 Specification . . . 21
4.3 Verifiable Credentials in re:claimID . . . 29
4.4 ReclaimPresentationSig2022 . . . 34
4.4.1 Linked Data Cryptographic Suites . . . 34

4.4.2 Canonicalization Problem 34
4.4.3 Specification 35
4.4.4 Example 36

5 Discussion **38**

6 Future Work **40**
6.1 Universal Resolver and Register 40
6.2 re:claimID Decentralized Identifier Library 40
6.3 Standard Compliant Linked Data Proof 41
6.4 OpenID Connect 41
6.5 DIDComm 41

List of Figures **42**

Bibliography **43**

1 Introduction

The term "Self-Sovereign Identity" (SSI) appeared in the 2010s. There is no formal definition for the term, rather it is an idea that is supported by different people and organizations. A SSI should always represent a real identity that exists in the real world. The SSI is at the center of the identity system and is in full control of and has access to all data related to his own identities, however, this does not mean it can change all claims about himself. A SSI may issue claims or facts about themself, which can be attested by other people, organizations, or the government. A person may have many different identities because they act differently in different social situations in which they are willing to share all or almost no information about themself.

To make the SSI a reality several organizations are working on different standards. We are going to explore two of the proposed standards, Decentralized Identifiers and Verifiable Credentials, and then map these to the GNU Name System and re:claimID. Both the GNU Name System and re:claimID are part of GNUnet. GNUnet is a peer-to-peer network stack that is secure, distributed, and respects users' privacy. "GNUnet's long-term goal is to serve as a development platform for the next generation of Internet protocols." [22i]

First, Decentralized Identifiers (DIDs): DIDs are globally unique decentralized, identifiers for individuals or objects that resolve to DID documents. DID documents mostly contain cryptographic information about the identity and answer some of the following questions: Which public keys should the identity use to authenticate itself? Can the identity issue Verifiable Credentials to other identities? There are many different methods that resolve DID documents. They all have their advantages and drawbacks.

Second, we will look at Verifiable Credentials. Verifiable Credentials are already widely used on the web. They use cryptographic proofs to attest one or multiple claims of a subject. The verifier of such a claim does not have to trust the subject, only the issuer.

In the following chapters, we are exploring the applicability of emerging identity standards to decentralized identity directories.

Chapter 2 provides background information about the new standards, GNUnet, and organizations working on the new standards.

Chapter 3 explains how others have implemented the Decentralized Identifier specification. First, the reference implementation which is based on the Bitcoin blockchain. Second, an implementation based on traditional TLS certificates trust assumptions and web infrastructure. Third, how blockchain projects such as Hyperledger fit into this.

Chapter 4 looks at how Decentralized Identifiers are implemented using the GNU Name System. It specifies the implementation according to the Decentralized Identifier specification where it analyzes its security and privacy. It then describes how W3C compliant Verifiable Credentials work in re:claimID.

Chapter 5 then discusses our findings and the problems we faced implementing the new standards.

Last, Chapter 6 lists work that could be done in the future.

2 Background

2.1 Decentralized Identifier

The Decentralized Identifier (DID) specification [Spo+21] describes DID's as follows:

> Decentralized Identifiers are a new type of identifier that enables verifiable, decentralized digital identity. A DID refers to any subject (e.g., a person, organization, thing, data model, abstract entity, etc.) as determined by the controller of the DID.

DIDs are a way to uniquely point to a person or object. They are decentralized if no central authority (expected of the controller) is in control of them. DIDs make use of cryptographic schemes, which make some actions or information about them verifiable to others. The subject of a DID is a person or an object described by the DID. While the controller of a DID is the party that is in control of the DID. E.g. a person controlling the DID of his car.

In Chapter 4 we are going to map DIDs to decentralized identity directories. We, therefore, explore the components of DIDs in the following sections. First, the syntax of a DID and how it is related to the Uniform Resource Identifier standard. Second, the methods used to interact with DIDs. A DID implementation has to provide a set of methods and follow certain guidelines. Third, the document a DID resolves to. DID documents contain information about DIDs. Fourth, the reasons for which DIDs are criticized.

2.1.1 DID Syntax

A DID is a Uniform Resource Identifier (URI). URI's are defined by RFC3986 [BFM05]. The URI standard gives a unique identifier to digital or physical resources. It does not define what exactly a resource is, although it gives examples as are files, people, or physical objects. The URI standard describes a general syntax that lets implementations parse the components of all URIs. Implementations do not have to understand all URI schemes to parse a URI. The specific URI schemes describe how each component should be interpreted.

A DID always begins with the *did* keyword, followed by the name of the DID method and an id defined by the DID method. The DID URI scheme is defined as followed:

```
did = "did:" method-name ":" method-specific-id
```

The DID URI scheme does not use the URI authority component and only consists of a path, query, and fragment. Listing 2.1 contains three valid DIDs using different DID methods.

```
did:web:example.com
did:btcr:xyv2-xzpq-q9wa-p7t
did:reclaim:000G0516V1NEP8NFZVJ2JHFGFS82PH94Q854FJCD6Q3FABAGEQ9CY3QHCC
```

Listing 2.1: Examples for valid DIDs

A DID is not only a unique identifier but also defines where the corresponding DID document can be found. A DID is a Uniform Resource Locator (URL). A DID is not a Uniform Resource Name (URN), because it is not location independent.

Figure 2.1 describes the relationships between URI, URL, and URN.

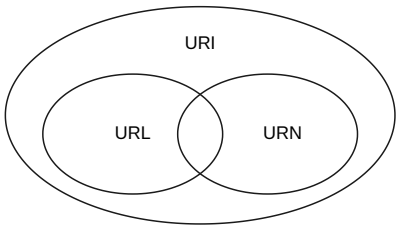


Figure 2.1: URIs, URLs, and URNs

It is possible to extend DID URLs to query specific parts of the DID document. A DID URL can contain query and fragment components similar to the HTTPS scheme.

```
did:btcr:xyv2-xzpq-q9wa-p7t/path
did:btcr:xyv2-xzpq-q9wa-p7t?query=test
did:btcr:xyv2-xzpq-q9wa-p7t#fragment-0
```

Listing 2.2: Examples for valid DID URLs

Using *#key-1* fragment with the DID document 4.1 would return the verification method

Inside a DID document, resources can be referenced using relative DID URLs. Relative DIDs omit the URI scheme, DID method-name, and DID method-specific-id components. They can contain a path, query, and fragment URI components.

The default reclaim DID Document 4.1 uses relative DID URLs.

In conclusion: DIDs follow the URI specification. This makes DIDs one of many other unique identifiers used on the Internet. DIDs are URLs because they describe where the corresponding DID document can be resolved.

2.1.2 DID Methods

The DID specification allows the creation of DID methods. A DID method has its own specification describing how to interact with the DID method. Many DID methods already exist using different underlying networks. The interface of DID methods is standardized which makes them interoperable. To make decentralized identity directories the underlying network for DIDs a DID method has to be specified and implemented. A DID method specification has to define multiple functions:

- How to **create** a DID and DID document.
- How to **resolve** a DID. The DID specification requires that the resolver can verify the authenticity of the response.
- How to **update** a DID document is an optional method. DID specifications either have to define how a DID document can be updated or state that updates are not possible.
- How to **deactivate** a DID and DID document is an optional method. DID specifications either have to define how to deactivate a DID and its associated DID document or state that deactivation is not possible.

Another requirement is that the DID method specification has to define how all methods authorize its execution. The specification needs to document “implementation considerations related to DIDs as well as Security and Privacy considerations” [Spo+21].

The specification defines 12 security consideration points of which 8 must be included in a DID method specification. A subset of which are going to be important for our DID method specification 4.2.1 is:
First, the specification must follow the guidelines of "Writing Security considerations Sections" [RK03]. Second, it must discuss how the DID method is protected against eavesdropping, replay, message insertion, deletion, modification, denial of service, amplification, and man-in-the-middle attacks. Third, describe the security risks by related protocols and risks of a compromised dependency. Fourth, explain how the DID method guarantees that each DID is unique. Fifth, define which data is to be held secret

(key material, random seeds, and so on) and label it clearly. Sixth, the specification should explain and specify the implementation of signatures on DID documents, if applicable. Seventh, if the DID method uses peer-to-peer computing resources, such as with all known distributed ledger technology, the expected burden of those resources should be discussed in relation to denial of service.

In addition, a DID method specification must discuss the privacy-specific threats described in "Privacy Considerations for Internet Protocols" [Coo+13]:
First, it must discuss the **Correlation** of different DIDs. Different DIDs of the same individual should not be correlating with each other. The DID specification advises using pairwise DID and DID document by default and only reusing DIDs when the correlation is wanted. DID documents should not contain the same verification method. Second, the specification must discuss the possibility of **Identification**. DIDs and DID documents should not link with the user's identity. DID documents are public records and should not contain identifying information. Verifiable Credentials are used to exchange personal information. Third, account for **Secondary Use**. Data should not be used for secondary purposes without the consent of the user. Fourth, discuss the **Disclosure** of the DID subject. The DID method should not reveal information about a user in a way that changes the way other people see the user. The threat of disclosure can stop a user from interacting because of the fear that their action might harm their reputation. Fifth, explain how **Exclusion** is prevented. Users should not be excluded in the decision-making about how other actors handle their data and have full knowledge about which information others have about them.

In summary: A DID method manages DIDs and DID documents. A DID is a unique identifier for an identity or an object. A DID document contains cryptographic information (e.g. public key) about the identity. A DID method specification needs to define the syntax of the DID, the methods to create, resolve, and optionally update and delete methods, and discuss possible security and privacy threats.

2.1.3 DID Documents

The DID document contains information about its subject. Most DID documents contain cryptographic public keys which make it possible for the DID subject to authenticate itself. A DID document can also contain methods for the attestation of Verifiable Credentials or for key agreement. The DID document does not contain information about the individual. DID documents are public records and can be read by anyone. An example of a DID document 4.1.

DID documents use JSON [17c] or JSON-LD [Spo+20] data format. While JSON is a widely used and known standard, JSON-LD is not. JSON-LD is a standard based on JSON that supports linked data. Linked data is also known as semantic web. It links data from different sources together and makes it readable to both machines and humans. The web today is very connected, a person browsing the web navigates it mainly by clicking links to other resources. Linked data uses this concept for data that is also "navigatable" by machines.

Most DID documents contain at least one assertion method. The assertion method may contain cryptographic material e.g. public keys. Verification methods can use different parameters e.g specifying a multi-signature scheme and are used for the following functionalities:

- **Authentication** of the DID subject
- **Assert** claims about themselves or others using Verifiable Credentials
- **Key agreement**: Tell others how an encrypted message can be sent to them e.g. using public cryptography.
- **Capability invocation**: Define how cryptographic capability can be invoked
- **Capability delegation**: Define how other DID subjects can use their cryptographic capability.

An example using multiple methods: A DID subject could authenticate to websites using a cryptographic key stored on the subject's smartphone. For the case, that the smartphone gets lost or the key gets compromised the DID subject could delegate the invocation capability either to a key stored on the subject's computer or to a cryptographic key in control by a trusted third party.

2.1.4 Criticism

The DID standard is criticized [21c] by Mozilla for multiple reasons:

- No practical interoperability: The Decentralized Identifier specification does not standardize any specific Decentralized Identifier method.
- Encourages divergence: The DID specification results in many different DID methods being used and therefore divergences.
- Centralized: The specification does allow centralized DID methods while the goal of the specification is to make identities decentralized.

- Proof-of-Work: The specification depends too much on the distributed ledger technology/ blockchains that require a lot of hashing power and need a lot of electricity often generated by burning fossil fuel.

2.2 Verifiable Credentials

Verifiable Credentials (VCs) [SLC21a] are used to share information, a credential, about the subject. A credential is verifiable when it contains a cryptographic proof for the information claimed in the credential. This does not mean that the information has to be correct, but assures that claims created by the issuer can not be tampered with. The subject and the issuer can be the same person. A credential where issuer and subject are the same person is a self-attested credential.

The amount of certainty a verifier puts into a VC depends on how much trust he puts into the issuer. If the verifier trusts the subject of a self-attested credential, it is as good as a credential issued by a trusted third party. VCs do not require DIDs and the DID standard does not depend on VCs however, they are often used together.

The following example 2.3 does contain all the required keys and is a valid VC. A VC can be extended by an *id* for the credential itself, *expiration data*, and *status* for whether the credential is suspended or revoked. It can also contain multiple credential subjects with multiple claims, e.g. a credential claiming the shared ownership of a physical object.

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
  ],
  "type": ["VerifiableCredential"],
  "issuer": "did:web:example.com",
  "issuanceDate": "2022-01-21T00:4:20Z",
  "credentialSubject": {
    "id": "did:btcr:xyv2-xzpq-q9wa-p7t",
    "name": "Satoshi"
  },
  "proof": {
    ...
  }
}
```

Listing 2.3: Examples for a Verifiable Credential

The proof contains cryptographic material attesting to the credential and making the credential temper resistant. The VC specification [SLC21a] does not define which kind of proofs are allowed, however, only different digital signatures are currently used and registered at the "Linked data cryptographic suite registry" [W3C20]

Figure 2.2 shows the trust relationships between the holder, issuer, and verifier and of a VC. It shows that the issuer gives a VC to the holder. The holder is responsible for storing the VC. He generates the VP and shows it to the verifier. The Verifiable Data Registry is explained in the next Section 2.3.

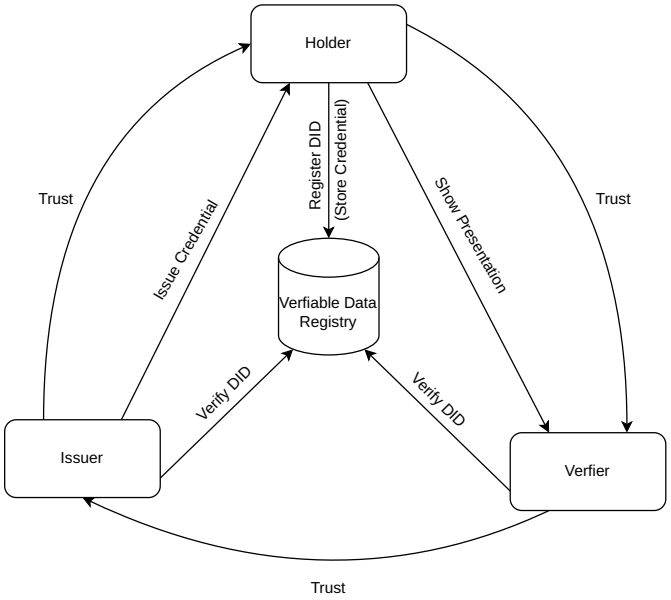


Figure 2.2: Actors in a Verifiable Credential scheme

2.2.1 Verifiable Presentations

VCs should not be shared directly. Whenever a user wants to share a claim he generates a Verifiable Presentation (VP). VPs are derived from VCs. VPs are presented when the subject shares one or multiple claims. A VP can increase the subject's privacy. Claims from multiple credentials can be combined in a single VP.

A simple VP contains a single VC and a proof created by the subject. The proof is a simple digital signature created by the subject. The subject shares all claims in the credential with the verifier even if the verifier only requested a single one.

The cryptographic scheme used to prove the validity made by the subject is not defined in the VC specification. Schemes supporting selective disclosure, complex combination of different VCs, reformating claims to the verifiers need, and zero-knowledge proofs hiding the credential itself is possible [SLC21b]

When using selective disclosure, the subject derives cryptographic proof from the credential and only shares a subset of claims from one or multiple credentials. How these proofs work in detail is out of scope. The "Linked data cryptographic suite registry" [W3C20] lists the BBS+ signatures scheme 2020 [21a] which can be used for selective disclosure if the Verifiable Credential is created using a BBS+ Signature [22o].

When multiple messages are signed using ECDSA a different signature is generated for every message. To prove that every message is signed, every signature must be presented. This is not the case for BBS+ signatures because it uses a paring-friendly curve. Paring-friendly curves allow the aggregation of signatures for different messages. The aggregated signature proves that all messages are signed. When a VC is created using a BBS+ signature every claim gets signed independently and the signatures get aggregated. Claims can be removed without breaking the signature of the credential. An example for selective disclosure: Alice wants to prove her real name to a verifier using a government-issued VC. The VC also contains information about Alice's residence and a picture of her. Alice's software uses selective disclosure to remove all information besides her name for the VP which is then presented to the verifier.

The subject can also choose to share claims derived from other claims using zero-knowledge proofs. This enables the subject to prove that some claim is true without leaking any additional information. The most used example is a subject proofs being over a certain age instead of sharing its birthdate. This is done by using a zero-knowledge range proof [00]. A zero-knowledge range proof is an algorithm that creates verifiable proof stating that a number issued by a trusted party is in between two other numbers without revealing the number itself. How this works in detail is out of scope.

Selective disclosure, zero-knowledge proofs, and other cryptographic schemes are important tools for a Self-Sovereign Identity. They protect user's privacy, while not compromising security nor convenience.

2.3 Verifiable Data Registry

The Verifiable Data Registry offers DID methods and may also store VCs. A DID method is often for a specific Verifiable Data Registry e.g. *did:btcr:xyz* 3.1 resolving DID documents stored on the Bitcoin blockchain. The Bitcoin blockchain is a Verifiable Data Registry. Verifiable Data Registries can use different underlying networks including peer-to-peer networks, distributed ledgers, any kind of centralized/federated client-server databases, distributed file systems, and decentralized identity directories.

Figure 2.2 shows different actors and how they can interact with the Verifiable Data Registry in the context of VCs. The holder of a VC and subject of a DID first registers a DID using the Verifiable Data Registry. The Issuer then verifies the given DID and issues a VC for the given DID. The holder can store the VC using the Verifiable Data Registry. When the verifier receives the VP he again verifies the VP using the DID stored on the Verifiable Data Registry.

Figure 2.3 shows the Verifiable Data Registry in the context of DIDs and VCs. The Verifiable Data Registry stores the VC, DIDs, and DID documents. It also shows the relationship between VCs and a DID. Note: DID controller and DID subject might be the same person or group. The subject does not have to be a person, it might also be a group or object. The DID controller might also be multiple persons, controlling the DID document.

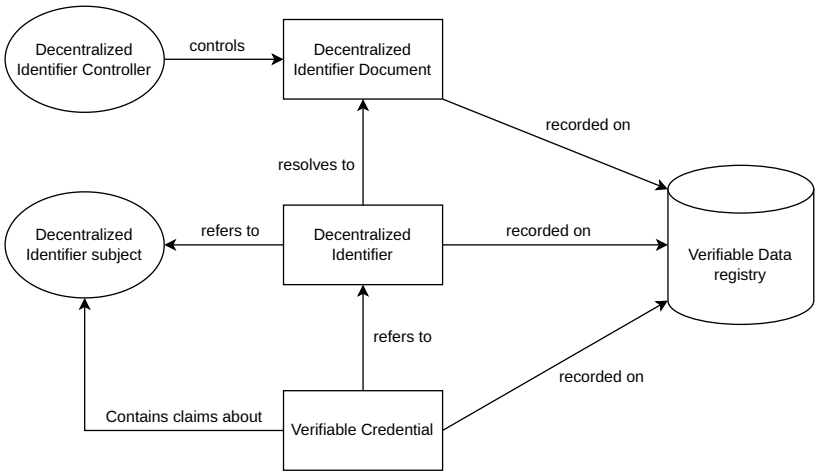


Figure 2.3: Actors and Components in a decentralized identity system

2.4 GNUnet

GNUnet is a collection of multiple peer-to-peer programs. Its goal is to provide a "global, distributed network that provides security and privacy" [22i]. GNUnet provides services for messaging, file-sharing, virtual private networks, Self-Sovereign Identity, and all the underlying dependency services. The following sections describe the three subsystems that are used to map the emerging identity standards (DID and VC) to a decentralized identity directory.

2.4.1 GNU Name System (GNS)

The GNU Name System [22p] is a decentralized, censorship-resistant, privacy-preserving name system. It is an alternative to the Domain Name System (DNS) [87].

Users can store records in zones that they are cryptographically in control of. GNS zones map one-to-one to public-private key pairs. Whoever is in control of a public key can add, delete, and change the records of a zone. Records in a GNS zone must be signed by the private key of a GNS zone and should have an expiration date after which they have to be renewed. Other peers in the network only propagate records that are signed by the private key of the zone. The public key is the top-level domain of a zone. Other users, who know the public key, can resolve records in that zone. GNS supports different kinds of records (e.g. TXT known from DNS).

While GNS can use different underlying storage layers, the GNUnet implementation utilizes a distributed hash table (DHT).

Zooko's triangle [01] is a trilemma of three wanted properties (Human-meaningful, Decentralized, Secure) in a naming system. GNS is fully decentralized in a way that only a person who knows a specific secret can create records for a specific zone. It is secure because no one else can change/delete that record. Zones however are not human-meaningful, they are the public keys of that zone, which needs lots of entropy to not collide with other public-private key pairs.

The following figure visualizes Zookos triangle and shows the difference between the GNU Name System and the Domain Name System (DNS). Globally unique names in DNS are human meaningful while they are not in GNS. GNS is decentralized and DNS is not.

If one wants to build a decentralized and secure DID method, they need the underlying Verifiable Data Registry to have the same properties. GNS makes the tradeoff of not being human meaningful but in return being as decentralized and as secure as possible for unique names. Self-Sovereign Identities most likely already have control over a public-private key pair, which can be mapped to a GNS zone.

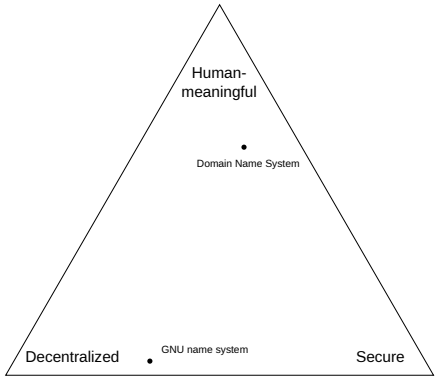


Figure 2.4: Zooko's triangle with GNU Name System and the Domain Name System

Therefore, GNS is a good choice to build a decentralized and secure DID method that supports the principles of Self-Sovereign Identity.

2.4.2 GNUnet Identity Subsystem

To use VCs and DID a user should have control over a public-private key pair. GNUnet Identity is a subsystem of GNUnet. It provides other services with cryptographic capabilities and handles the private keys of users. GNUnet Identity provides other services with the capability to sign and verify arbitrary data and the other services can encrypt and decrypt arbitrary data. Self-Sovereign Identities should have control over at least one public-private key pair. All current authentication methods used by Decentralized Identifiers use some kind of public-key cryptography.

In GNUnet Identity, a user has control over identities called egos. A user can have multiple egos of which each has its public-private key pair. Egos can represent arbitrary things, e.g. the user's identity, pseudonyms, or an organization managed by the user. One principle of Self-Sovereign Identity is that a user can multiple identities for different social situations. It is fitting that GNUnet Identity can already support multiple identities.

GNUnet Identity has two types of elliptic curve digital signature algorithm. The standard ego uses ECDSA with Curve25519 [FIS05]. An alternative is EdDSA which uses the edwards25519 curve [17a].

2.4.3 re:claimID

re:claimID is a Self-Sovereign Identity provider (IdP). An identity provider is a service that creates, manages, and delivers identity data to other services.

Some of the biggest identity providers right now are Google, Facebook, and Apple. These companies serve as identity providers and manage a user's identity. They help a user to create a profile on a third party's website and make it easy for a user to change information about themself, without having to manage every single website. These companies are trusted third parties. A user has to trust them and give up his privacy for convenience. The base concept of Self-Sovereign Identity (SSI) is that the user is at the center of the management of his identity. They should be in control and their privacy should be protected. This is where re:claimID comes into play: re:claimID can deliver the same convenience to users while being in line with the principles of SSI. A user can add information about themself and also import attested information from other parties. The next time a service needs personal information from the user, they can share parts of the information about them managed by re:claimID without relying on a third party and without leaking any data about the process to anyone.

In re:claimID each ego (managed by GNUnet Identity) can store attributes. Attributes describe the ego (e.g. name, email address, date-of-birth). Attributes are saved encrypted under a random name in the ego's GNS zone. The encryption key is derived from the random name and the namespace's public key. It is not possible for someone who does not know the random name to find and read the record. The user can share one or multiple attributes with a requesting party. When a user wants to share one or multiple attributes he creates a new namespace with a random name. In this new namespace, he then creates mappings from the keys of the attribute to the random name under which the attributes are stored in GNS. He then transfers the ownership of the namespace to the requesting party. The requesting party is then able to follow the mappings to the value of the attributes. If the user wants to remove access he moves the record storing the attribute to a different random name. The transferred namespace now maps to records that no longer exist.

2.5 Organisations working on new Identity Standards

The **World Wide Web Consortium (W3C)** is by far the biggest organization working on Self-Sovereign Identity standards. It created the DID [Spo+21] and the VC [SLC21a] specification. The W3C Credentials Community Group [22a] has several projects [22b], supporting the new standards created by W3C. Some of them are: DID method registry, JSON-LD cryptographic proof registry, Web DID method, Bitcoin DID method, specification for JSON-LD BBS+/JWS/EdDSA/ECDSA/RSA signature suits, test suits for DID methods, multi-base format, and many educational resources.

In its vision of the future, [22q] the W3C claims to support the Web-of-Trust and support users privacy.

The **Decentralized Identity Foundation (DIF)** [22c] exists to develop decentralized identity technology. DIF supports the W3C standards and their projects are built on top of them. The DIF is a collaboration between organizations and individuals that all support the goal of the self-sovereign identity that is interoperable and open source. Some of the DIF work [22d] is the Universal Resolver/Register which can resolve/create DID document for many different DID methods. Libraries for creating and verifying VCs and VPs, and libraries supporting DIDComm exist in different languages. The JSON-LD PGP signature suite and resolvers for different DID methods in JavaScript and Java [22d] are being developed.

The goal of the **OpenID Foundation (OIDF)** [22l] is to support the OpenID community and promote the OpenID standard. OpenID Connect is used by many big websites and services. For example Google, Facebook, Microsoft, Apple, Yahoo, PayPal, Amazon, GitHub, and more are providing authentication services via OpenID connect as identity providers.

While the OpenID standard is not self-sovereign identity technology the OpenID Connect workgroup specified the Self-Issued OpenID provider (SIOP). SIOP makes it possible for users to directly authenticate to a service using the OpenID Connect standard. Since many big websites and services use OpenID Connect with traditional identity providers (e.g. Google, Facebook, Apple), SIOP would allow users to be their own identity providers using emerging identity standards without big changes to the underlying infrastructure.

OpenID connect supports W3C VCs using an extension [Ter+22]. The extension allows the transmission of VCs via OpenID Connect. This plugin is independent of the Self-Issued OpenID provider but when combined it makes it possible to use the new identity standards via the existing OpenID Connect infrastructure.
Both together makes OpenID Connect compatible with Self-Sovereign Identity principles

Rebooting the Web of Trust (RWTO) [22n] is a yearly workshop first, hosted by Christopher Allen, and supports the development of Self-Sovereign Identity. The workshop pushes the development of Self-Sovereign Identity and tries to unite different developers. They publish papers [22r] on various Self-Sovereign Identity topics, mainly about DIDs, VCs and the web of trust.

All organizations are supporting the standards purposed by the W3C and there is no competing standard with similar support. All organizations have the common goal to protect user's privacy. While the W3C and OpenID Foundation are large organizations that follow multiple goals, the Decentralized Identity Foundation, and Rebooting the Web of Trust focus almost exclusively on new identity standards. One difference is that the W3C designed the DID standard that everything (including objects) can have a DID, while the DIF and RWTO focus on the concepts of Self-Sovereign Identity (only persons). One of the core principles of SSI is that it should always represent a real human identity.

3 Related Work

The following three sections describe two existing DID methods and Hyperledger Aries. First, a DID method based on the Bitcoin blockchain. Second, a DID method using existing web infrastructure and third how the Hyperledger project is related to DIDs and VCs.

3.1 Bitcoin DID Method

The Bitcoin DID method (did:btcr:) [All+19] is the reference implementation for the DID specification. The specification of the Bitcoin DID method is an example for other implementation specifications. As described in the BTCR specification the method is a "DID trust anchor based on the Bitcoin blockchain, updates publicly visible and auditable via Bitcoin transactions, and optionally, additional DID Document information referenced in the transaction OP_RETURN data field" [All+19]. The BTCR DID method should serve as a very secure example for other DID methods.

Bitcoin transactions: A Bitcoin transaction consists of inputs and outputs. Transaction inputs are the transaction outputs of other transactions. The Bitcoin blockchain is a series of transactions that starts with a special transaction created by a miner. Each transaction output has a locking script. Whoever can fulfill the script can spend an output in another transaction as input. The transaction output also contains the amount of Bitcoin that is spendable with that output.

Bitcoin scripting language: The Bitcoin scripting language consists of opcodes, of which some perform cryptographic functions. A very simple example, taken from the book "Mastering Bitcoin" [17b], of a locking script looks like this:

```
3 OP_ADD 5 OP_EQUAL
```

When creating a transaction one has to reference at least one transaction output and provide a script that contains Bitcoin opcodes. The provided opcodes are put in front of the locking script. The transaction output can be spent should the script evaluate to true. Adding 2 to our example script evaluates to True.

```
2 3 OP_ADD 5 OP_EQUAL
```

The OP_RETURN opcode is special. Any transaction output containing it is unspendable and any value attached to a transaction output containing it will be burned. The Opcode can add 80 bytes of arbitrary data to a transaction, while still being a valid transaction. The Opcode was created in 2013 as a compromise because more and more people created unspendable transaction outputs to store all kinds of data. This led to the growth of the set of unspent transaction outputs (*UTXOs*), which is held in memory by the Bitcoin Core client. Using OP_RETURN people are still able to store arbitrary data on the Bitcoin blockchain while not growing the set of *UTXOs*.

Bitcoin transaction reference: A Transaction reference *Txref* is a reference to a Bitcoin transaction. It contains a reference to the chain, block, transaction index, and optionally the output of a transaction. It is also part of every transaction input.

The Bitcoin DID is a reference to a transaction using the *Txref*.

$$\text{BTCR DID} = \quad \text{"did:btcr": } \textit{Txref}$$

When a DID is created, the subject sends a transaction to a public key it is in control of. It contains an optional second transaction output containing an OP_RETURN opcode followed by a URL that has to resolve to a DID document. If the optional second transaction output is not used, a standard DID document has to be created by the resolver. The BTCR DID document is very similar to the default re:claimID DID document (the reclaim:ID DID document was inspired by BTCR). The only difference is that the BRCR standard DID document has two copies of the same verification methods. One for authentication and one for assertion. The public key inside the verification method is derived from the Bitcoin address that sends the transaction.

When a BTCR DID is resolved the client checks if the transaction output is spent. If the output is not spent and the transaction does not contain a second transaction output, the client generates the standard DID document based on the public key and *Txref*. If it contains a second output the URL is resolved. Should the first transaction output be spent the client checks the transaction that spent that output.

To update the DID document of a BTCR DID, the user spends the first transaction output and constructs the transaction as he would have the first time. However, all updates must contain a second output with an OP_RETURN opcode pointing to a URL.

The DID document is deleted if the subject spends the last transaction output without adding a second transaction output containing an OP_RETURN Opcode.

3.2 Web DID Method

The Web DID method (did:web:) [Gri+21] is different from most other DID methods. It does not rely on any distributed ledger technology or any other peer-to-peer system. Instead, it uses traditional domain names and a client-server model to serve DID documents. DIDs in the DID web method inherit trust put into an already known domain name. A DID in the web method is built as followed

web-did = "did:web:" domain-name
web-did = "did:web:" domain-name * (":" path)

Example DID looks like this:

```
did:web:w3c-ccg.github.io
did:web:w3c-ccg.github.io:user:alice
did:web:tristan.schwieren.xyz
```

In order to create a DID, the subject must have control over a domain pointing to a web server that can serve files using HTTPS with at least support for TLS1.2. The DID document must be served by the web server under a specific path.

The following examples explain how DIDs are converted to URLs. ':' are replaced with '/'; "did:web:" is replaced with "https://"; If the path is not relative then "/.well-known" has to be added otherwise "/user"; "/did.json" is appended at the end. Note: resolve the last URL using a web browser or *wget*.

did:web:w3c-ccg.github.io
-> https://w3c-ccg.github.io/.well-known/did.json
did:web:w3c-ccg.github.io:user:alice
-> https://w3c-ccg.github.io/user/alice/did.json
did:web:tristan.schwieren.xyz
-> `https://tristan.schwieren.xyz/.well-known/did.json`

In order to resolve a web DID the resolver must make an HTTPS (at least TLS1.2) GET request to a web server. The DID can be updated by changing the served DID document and can be deleted if the server stops serving the DID document.

3.3 Hyperledger Aries and Indy

Hyperledger Aries is an infrastructure project. This means it is not a standalone project but a set of libraries and other programs for other projects.

It provides a set of tools for distributed ledger technology to be interoperable with new identity standards. Hyperledger Aries is not a blockchain, p2p network, or Verifiable Data Registry, it is a bridge between them. Hyperledger Aries includes multiple libraries [22k]:

First, a communication protocol for DID names **DIDComm** [Fou21a]. It allows the communication between DIDs over multiple types of transport (HTTP, email, signal, QR codes, NFC, Bluetooth, etc.). Communication channels do not have to provide any kind of cryptographic assurances because that is handled by the DIDComm protocol. Second, **Resolvers** that implement a DID methods. They can resolve DIDs for DID documents. Initially, Hyperledger Aries only supports a single resolver for Hyperledger Indy but Hyperlegder Aries is built to support all DID methods. Third, a **Wallet** providing cryptographic functionality and secure storage of key material and Verifiable Credentials. Fourth a library providing **Zero-knowledge capability**. Hyperledger Aries provides W3C Verifiable Credential compatible zero-knowledge libraries. It can be used to create selective disclosure Verifiable Presentations.

Hyperledger Aries focuses on using distributed ledger technology (Hyperledger Indy and Ethereum) for Self-Sovereign Identity applications. However, its goal is to be interoperable with other DID methods and follow W3C standards for Verifiable Credentials [HB21].

4 Mapping Self-Sovereign Identity Standards to re:claimID

4.1 Approach

We approach the mapping of the previously explained new identity standards (Decentralized Identifier and Verifiable Credentials) as follows: We first implemented the Decentralized Identifier on top of GNUnets GNU Name System implementation. We then have written a DID method specification according to the guidelines. We created a plugin for re:claimID that supports W3C Verifiable Credentials [SLC21a].

4.2 re:claimID Decentralized Identifier Method

We implemented a DID method using the GNU Name System (GNS) and GNUnet Identity. GNS serves as the Verifiable Data Registry while GNUnet Identity handles the cryptographic functionalities. The proof of concept implementation can be found on GNUnet's git [22h]. The implementation is a proof of concept and not a DID method ready for production. It does serve all the necessary functionalities of DID method. In the following, we are specifying our DID method implementation as required by the Decentralized Identifier Specification [Spo+21].

4.2.1 Specification

We are specifying the re:claimID Decentralized Identifier (DID) Method. The Bitcoin DID Method (named *BTCR*) is the reference implementation for the DID standard and the Bitcoin DID specification [All+19] conforms with the requirements specified in the DID specification [Spo+21]. We, therefore, use the BTCR DID Method as our template.

Introduction The re:claimID DID method adds supports for the DID standard in re:claimID [22m] which is a service on GNUnet [22g].

Terminology

GNU Name System is a naming system that is decentralized and secure. Users can register zones and resolve the names of other users. Zones can be registered by anyone, users do not need permission to do so. Zones are not human-rememberable. The name of a zone is the public key of the ego controlling it.

Egos control a GNS namespace. Each ego has a single public-private elliptic curve key pair. An ego should not be confused with a user. A user can have multiple egos.

Basic Concept The re:claimID DID method utilizes the GNU Name System (GNS) and egos. It stores DID document in the root GNS zone of an ego, using the GNUnet Namestore service. The security, therefore, depends on the security of the GNU Name system.

re:claimID DID Format The DID contains the *did* and *reclaim* keyword and the public key of the ego encoded with base32gns [22p]. They have the following format:

re:claimID DID =	"did:reclaim": ego-pubkey
	[did-fragment]
ego-pubkey =	EdD25519 public key of the ego encoded using base32gns

re:claimID DID construction re:claimID DIDs are only dependent on the ego's public key. When a DID document is created a new ego is also created. *gnunet-did* then gets the egos public key from the GNUnet identity service and constructs the DID.

Default DID document When a user creates a new ego without any optional arguments a, standard DID document is created using the newly generated DID. The DID document contains a single *verificationMethod* that uses the public-private key pair of the ego. The multi-base [BS22] encoding of the egos public key is encoded in base64. GNUnet supports two types of digital signature algorithms, but egos created by *gnunet-did* always use EdDSA keys. The idea behind this decision was to support JSON-LD proof Ed25519VerificationKey2020 [Sab21]. The ECDSA implementation of GNUnet uses Curve25519 [22i] while the only ECDSA linked data proof uses Secp256k1 [Bro10].

Note: the *ego-pubkey* is shortened in the following example.

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/ed25519-2020/v1"
  ],
  "id": "did:reclaim:000G050N7WJYG7R...D4WGXP0ND7TE8HW8",
  "verificationMethod": [
    {
      "id": "did:reclaim:000G050N7WJYG7R...D4WGXP0ND7TE8HW8#key-1",
      "type": "Ed25519VerificationKey2020",
      "controller": "did:reclaim:000G050N7WJYG7R...D4WGXP0ND7TE8HW8",
      "publicKeyMultiBase": "u7QEVPyXo...tgVafTkR4g"
    }
  ],
  "authentication": [
    "#key-1"
  ],
  "assertionMethod": [
    "#key-1"
  ]
}
```

Listing 4.1: A default DID document

Operations

Creating a DID When a user creates a new DID and its associated DID document the *gnunet-did* program first creates a new ego using the GUNuet Identity service. It then uses the public key of the returned ego to generate the DID after the previously described format 4.2.1 and generates the default DID document 4.1. The new DID document is sent to STDOUT. GNUnet Namestore is used to store the DID document as a TXT record in the root of the egos GNS zone using the empty label @. It takes some time until the new record propagates through the GNU Name System.

In sequence diagram 4.1 Alice is creating a DID/-document for herself.

Reading a DID To read a DID document the user needs the associated DID. The DID is resolved. A DID contains an ego's public key. The public key of an ego is the zone in the GNU Name System. It is the zone an ego is cryptographically in control of. Since a DID document must be stored in the root zone with an empty label we know exactly where it is stored. Querying GNS for the extracted public key returns the DID document. GNS might return multiple records and even multiple TXT records if the ego stored other things there. This is not handled in our proof of concept implementation.

At the bottom of sequence diagram 4.1, Bob resolves the DID given by Alice. Alice first asked *gnunet-did* for her DID but she could also extract her DID from the returned DID document which was printed earlier. She then shares her DID with Bob. How she shares the DID with him is irrelevant for this specification. Bob then sends a resolve request with the given DID to *gnunet-did*. *gnunet-did* then extracts Alice's public key and requests Alice's root zone to GNS. *gnunet-did* prints the returned TXT record to STDOUT. The returned DID document looks like example 4.1.

Alice is now able to authenticate herself to Bob assuming she is using the standard DID document or a custom DID document generated by her which also contains an *authentication* method.

Removing a DID When a user removes a DID and its associated DID document the *gnunet-did* program first looks up the given ego using the GNUnet Identity service. *gnunet-did* then uses the GNUent Namestore program to remove the DID document. It takes time for the removal to propagate through GNS. A peer in GNS can not enforce the removal of his records. Alice can not verify that every node in GNUnet removed her record and can not stop peers from serving the record on request, some nodes might still save/serve it. The record however becomes invalid to everyone after a pre-set time.

Updating a DID When a user updates his DID document *gnunet-did* first performs a DID remove and than a DID create.

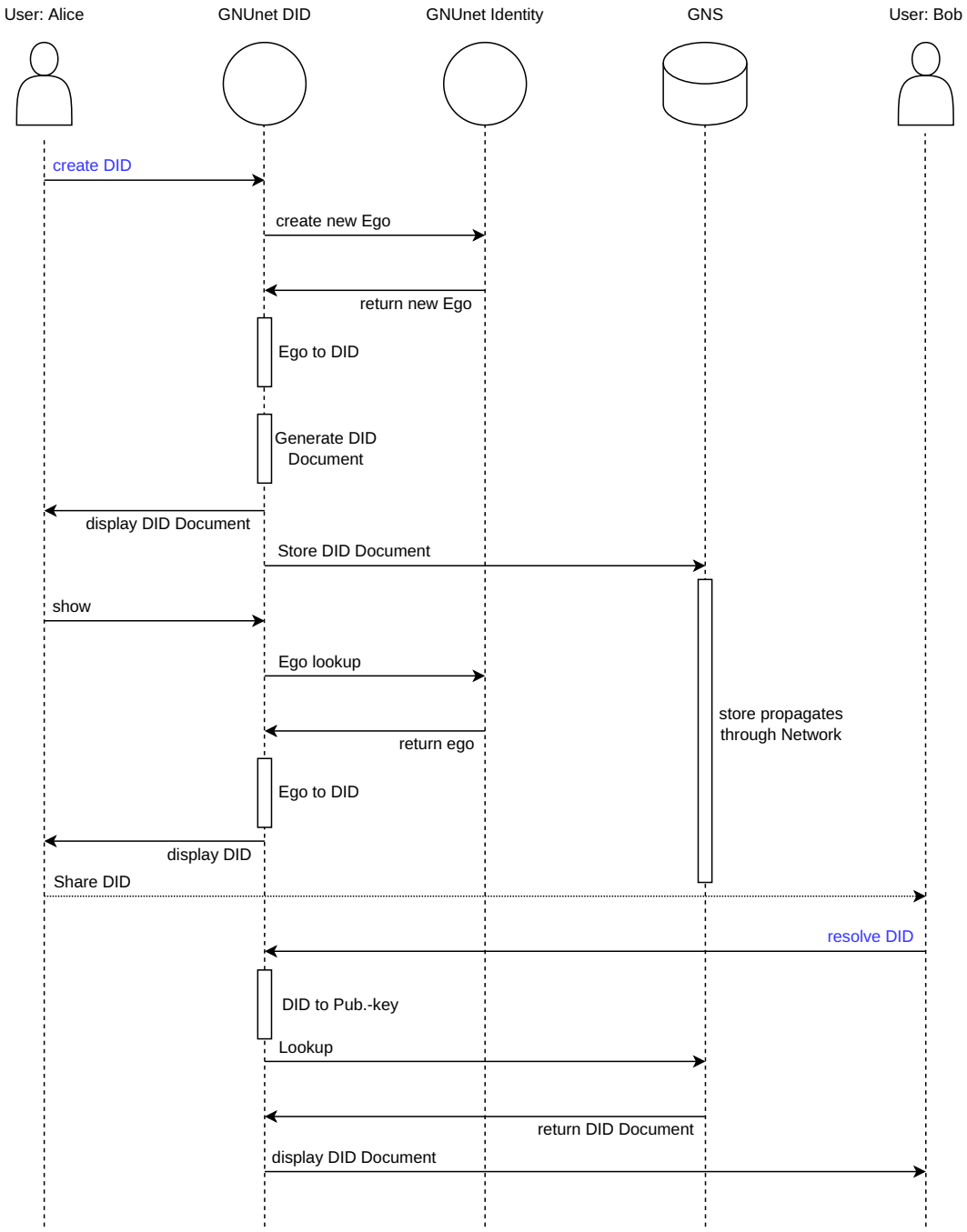


Figure 4.1: Creating and resolving a DID (document)

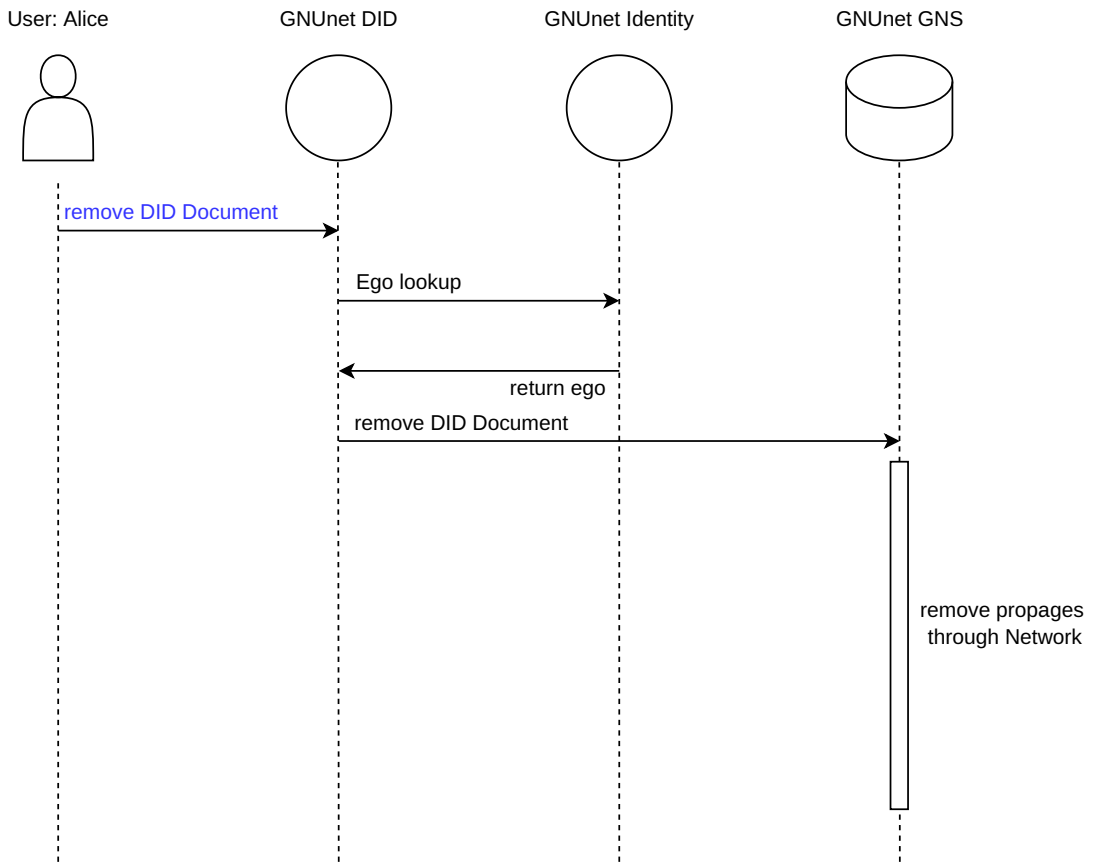


Figure 4.2: Removing a DID (document)

Security considerations In general, *gnuent-did* inherits the security of GNUnet, especially the security of the GNU Name System. In the following, we answer the security considerations listed in the Decentralized Identifier specification [Spo+21]. Note: The question itself is written bold, followed by the answer.

The Security Considerations section MUST document the following forms of attack for the DID operations defined in the DID method specification: eavesdropping, replay, message insertion, deletion, modification, denial of service, amplification, and man-in-the-middle. Other known forms of attack SHOULD also be documented. The DID method is not susceptible to an eavesdropping attack since no transmitted data needs to be kept secret and the private key is reasonably protected by GNUnet Identity. No secrets are transmitted over GNS, therefore replaying a message does not make sense. A malicious peer could transmit an already invalidated DID document, however, peers ask multiple peers for revocation. The DID method can not be attacked by insertion or modification attacks. An attacker can not return a manipulated/made-up DID document since the record would not be signed by the correct key. An attacker can only delete records on his peer and not the records stored by others. GNUnet is protected against Man-in-the-middle attacks using an address validation protocol [22i]. GNUnet can not be attacked with an amplification attack because this requires the attacker to impersonate another peer which is not possible if it does not know the victim's private key. The underlying distributed hash table of GNS is protected against denial of service attacks [22i].

The Security Considerations section MUST discuss residual risks, such as the risks from compromise in a related protocol, incorrect implementation, or cipher after threat mitigation was deployed. An incorrect implementation of the digital signature algorithm in GNUnet Identity could make it possible for an attacker to impersonate any other ego, and create/delete DID/-documents. An error in the GNS implementations could have the same effect. GNS has crypto-agility. This means that GNS can switch to a different cipher should the current one be broken [22p].

The Security Considerations section MUST discuss the policy mechanism by which DIDs are proven to be uniquely assigned. DID's are unique because they contain the public key corresponding to the user's GNS zone. The chance for two users to generate the same private key and derive the same public key is negligible.

If a protocol incorporates cryptographic protection mechanisms, the DID method specification MUST clearly indicate which portions of the data are protected and by what protections, and it SHOULD give an indication of the sorts of attacks to which the cryptographic protection is susceptible. Some examples are integrity only, and endpoint authentication. The DID method uses digital signatures. The security of the DID method depends on the assumption that a user can keep his private key secret. The private keys are managed and kept secret by GNUnet Identity.

Data which is to be held secret (keying material, random seeds, and so on) should be clearly labeled. The specified method never displays the private key of the DID. The user can however display the DIDs private key using GNUnet Identity.

DID method specifications should explain and specify the implementation of signatures on DID documents, if applicable. DID documents are not signed directly. DID documents are stored in the root of GNU Name System zones. Every record in a user's zone is signed by the user's private key.

Where DID methods use peer-to-peer computing resources, such as with all known DLTs, the expected burdens of those resources SHOULD be discussed in relation to denial of service. The underlying storage layer used by the DID method is a distributed hash table. Distributed Ledger Technology is not used. GNUnet and GNS still need resources (for relaying messages, storing other peers records, etc.), however, a peer can not be forced to provide these resources. A peer could use more resources than it provides.

Privacy considerations In the following, we answer the privacy considerations for a DID specification described in Section 2.1.2: **Correlation:** Two DIDs can not be correlated. Every DID uses its own private-public key pair that is randomly generated. Privacy can be compromised if the user decides to use a custom unique DID document. **Identification:** The user can not be identified using his DID. The DID doesn't contain any identifying information. **Secondary Use:** The use can not prevent others to share his DID and DID document. They are public records. **Disclosure:** The DID method does not reveal any information that could harm the users reputation. **Exclusion:** Users only reveal their DID document. However, the user has no control over how others handle that data.

Usage Example We are now looking at some examples of how Alice and Bob would use *gnunet-did*:

```
$ gnunet-did --create --ego=alice --expiration-time=1d
```

Returns a DID document such as 4.1 and stores the DID document in GNS

```
$ gnunet-did --ego=alice --show
did:reclaim:000G0516V1NEP8NFZVJ2JHFGFS82PH94Q854FJCD6Q3FABAGEQ9CY3QHCC
```

displays Alice's DID.

```
$ gnunet-did --show-all
did:reclaim:000G0516V1NEP8NFZVJ2JHFGFS82PH94Q854FJCD6Q3FABAGEQ9CY3QHCC
did:reclaim:000G053J77ZHZAN99J5PMFYSV247ER1GED0JP9QE1EWMB80YZZCMP5PXEW
did:reclaim:000G0525CKX4Y6RCCCNSKG6ENATKQDB0QF9YYJCZEHC1X05XA689SNAQFG
did:reclaim:000G056BS52SWE7B8FWTNRG05YVCZJ7C8N2ZP8EYMEVGWYX6SGQMKCC9B0
```

displays all DID's of ego's controlled by the user that have a DID document.

```
$ gnunet-did --ego=alice --remove
```

deletes the DID document

```
$ gnunet-did --get --did=did:reclaim:000G0516V1NEP8N...FABAGEQ9CY3QHCC
```

Returns a DID document such as 4.1

4.3 Verifiable Credentials in re:claimID

Support for W3C Verifiable Credential in re:claimID is achieved through a credential plugin. The implementation [22j] supports a simple Verifiable Credential, containing one or more claims about a single subject. It assumes the subject is using the standard re:claimID DID document or a DID document having at least the same capabilities. It has no protection against replay attacks (challenge-response field). The proof to verify the presentation is not compatible with other implementations. We should also note that the implementation does not implement JSON-LD Verifiable Credentials but is compatible with them using a simple preset presentation template.

re:claimID supports different kinds of attributes. The normal attributes are simple self-attested attributes. More attributes can be added using a plugin interface. Plugins implement a credential type. Already implemented plugins support JSON Web Tokens [15] and Privacy-preserving Attribute-based Credentials [21b]. Plugins have to implement a set of functions to re:claimID. Several getter functions for different attributes of Verifiable Credentials/Presentations and most importantly a function for generating the Verifiable Presentation.

The Verifiable Presentation is generated as followed:

1. Use template for the *context* and *type* key-value pairs.
2. Import the entire Verifiable Credential under the *verifiableCredential* key
3. Set *created* in *proof* to current time.
4. Set *verificationMethod* in *proof* to subject DID with relative reference to *key-1*.
5. Give the entire presentation (with empty *signature*) to ReclaimPresentationSig2022 algorithm
6. Set the returned signature in the *signature* field.

The following diagram shows the relevant, different components and their dependency when using re:claimID with plugins.

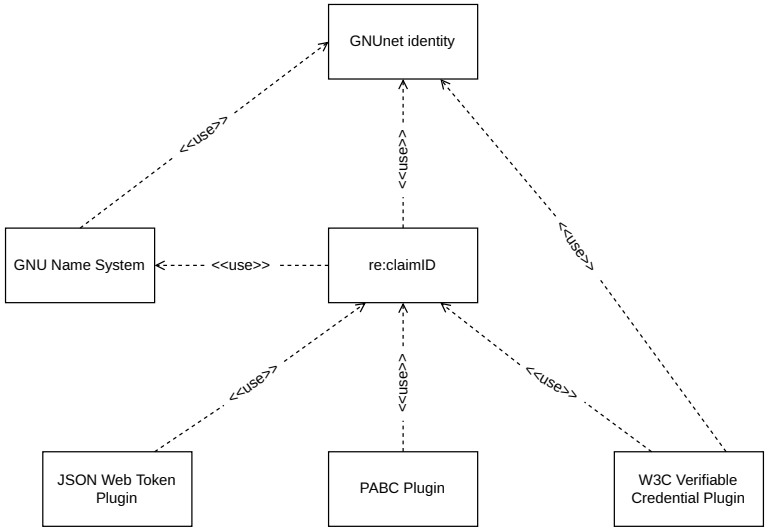


Figure 4.3: Dependencies between re:claimID, GNS, GNUnet Identity and plugins

The user Alice wants to store a Verifiable Credential in re:reclaimID. Bob later requests attributes from the Verifiable Credential from Alice. The process goes as followed:

1. Alice calls re:claimID as seen in 4.3.
2. re:claimID looks up the given ego and returns it.
3. re:claimID encrypts the given credential and stores it under a random name in GNS
4. re:claimID creates attributes based on the credential
5. Bob requests attributes from Alice
6. Alices creates a ticket for Bob and calls re:claimID as described in 4.3
7. re:cliamID generates a Verifiable Presentation and stores it under a random name in GNS.
8. Alice generates a ticket that contains the random name under which the presentation is stored.
9. Alice sends the ticket to Bob.
10. Bob gives the ticket to re:claimID
11. re:claimID knows the name under which the presentation is stored from the ticket and resolves it.
12. re:cliamID decrypts the record.
13. re:claimID displays the Verifiable Presentation to Bob.

The following sequence diagram shows how a user can add a Verifiable Credential and share a Verifiable Presentation with another user:

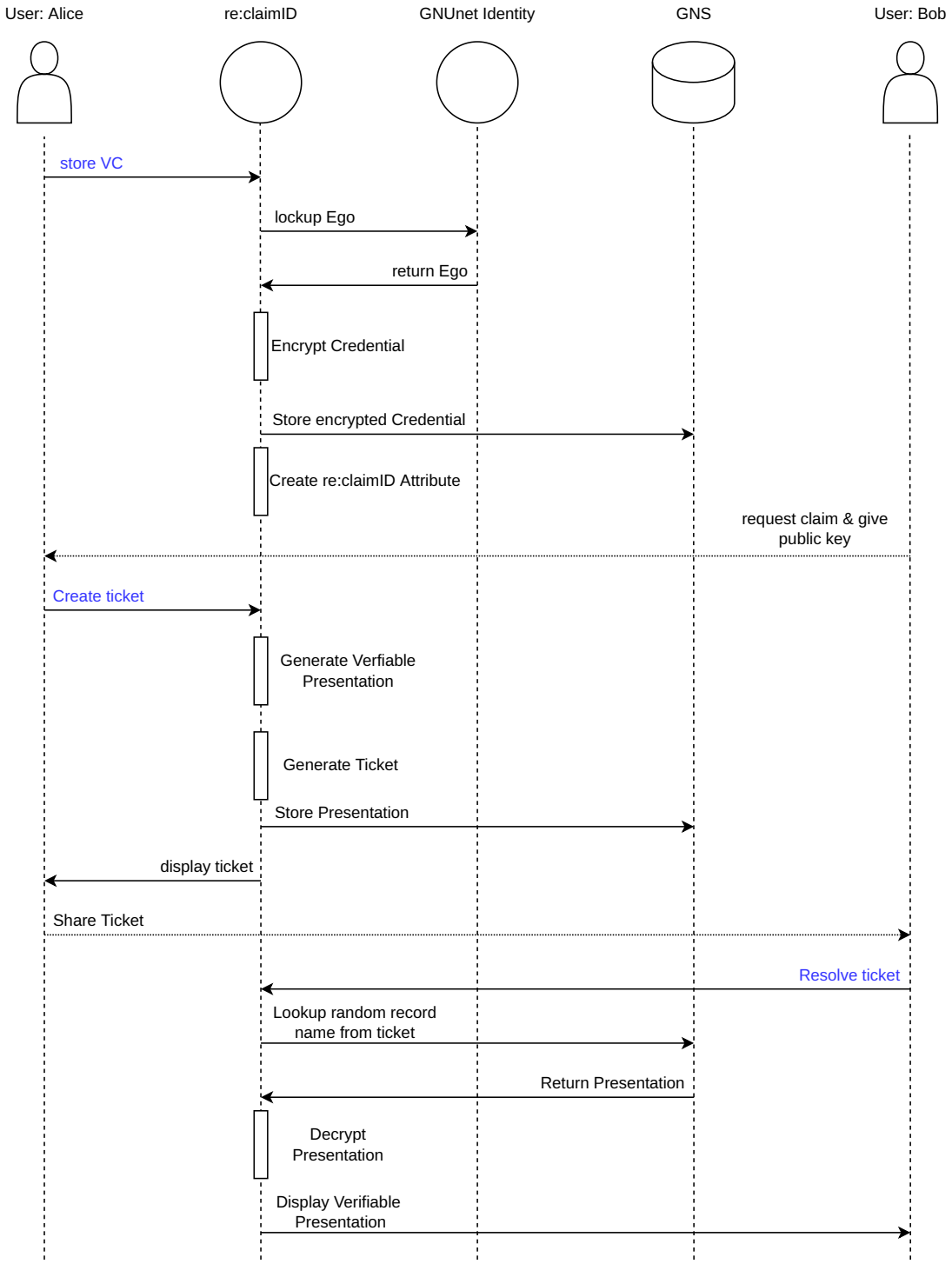


Figure 4.4: Adding a Verifiable Credential to re:claimID and sharing a Verifiable Presentation

An expample of a Verifiable Credential that can be used with GNUnet looks as followed:

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1"
  ],
  "type": [
    "VerifiableCredential"
  ],
  "issuer": "did:web:bobs-blog.xyz",
  "issuanceDate": "2022-01-26T19:00:00Z",
  "expirationDate": "2025-01-26T19:00:00Z",
  "credentialSubject": {
    "id": "did:reclaim:000G0516V1NEP8N...FABAGEQ9CY3QHCC",
    "name": "Alice"
  },
  "proof": {
    "type": "RsaSignature2018",
    "created": "2022-01-26T19:00:00Z",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "did:web:bobs-blog.xyz#key-0",
    "proof": "2SWE7B8FWTNRG05...GWYX6SGQMKCC9B0"
  }
}
```

Listing 4.2: A simple Verifiable Credential

A user uses the Verifiable Credential plugin as followed:

```
# Import Credential
$ gnunet-reclaim --ego=alice --credential-name=cred1 \
--credential-type=VC --value=$vc

# Create Attribute based on Credential
$ gnunet-reclaim --ego=alice --add=cred1 --value=name \
--credential-id=$cred1_id

# Issue a ticket for the attribute
$ gnunet-reclaim --ego=alice --issue=cred1 \
--rp=000G0525CKX4Y6RCCCNSKG6ENATKQDB0QF9YYJCZEHC1X05XA689SNAQFG
```

```
# Consume Ticket
$ gnunet-reclaim --ego=charly -C $ticket
```

Listing 4.3: Importing a Credential; Creating & issuing a Presentation in re:claimID

4.4 ReclaimPresentationSig2022

4.4.1 Linked Data Cryptographic Suites

Linked Data proofs are cryptographic proof for the JSON-LD data format. They provide Verifiable Credentials and Verifiable Presentation with authenticity and protect the integrity. Different types of proof exist, most of them being digital signatures. Some proofs use zero-knowledge technology. Zero-knowledge proofs are mainly used for selective disclosure or range proofs in Verifiable Presentations. Only showing a subset of claims, and proofing a value (e.g. age) being in a certain range.

A digital signature linked data cryptographic proof consists of multiple parts:

- **Canonicalization** Algorithm: Canonicalization is the process of deterministically converting the same information in different ways to the same format.
- **Hashing** Algorithm: The Hashing algorithm deterministically reduces any kind of data to a small representation of that data. Hashes are of the same length and most of the time change significantly on small changes of the input data.
- **Signature** Algorithm: Take an input message and produce a digital signature so that a receiver of the message can verify the integrity of the message and be sure that the singer has a specific secret.

Linked Data Proofs first canonicalize the information, then hash the standardized representation, and after that sign the hash. This proves that the information in the Linked Data was created by the signer and the information can not be manipulated without breaking the signature.

4.4.2 Canonicalization Problem

Several standardized Linked Data Proofs such as *Ed25519 Signature 2018* [Sab21] already exist and are well supported. The problem is that there is no implementation for the canonicalization algorithm in C. Canonicalization is needed before hashing data. Without canonicalization, two credentials/presentations containing the same information would produce different hashes when formatted slightly different.

The canonicalization algorithm for JSON-LD is anything but trivial and in general, seemed unfinished to us. We realized that it would be out of scope to implement the algorithm ourselves and created our simple JSON-LD proof.

4.4.3 Specification

The ReclaimPresentationSig2022 provides two functions: one for generating signatures and one for verifying a message.

The signature generation function expects a JSON containing a Verifiable Presentation without the signature key-value pair in the proof section. Note: A Verifiable Presentation contains two proofs. One is created by the issuer of the Verifiable Credential which protects the authenticity and integrity, the presentation proof is created by the controller of the DID.

```
{
  {
    "@context": [
      "https://www.w3.org/2018/credentials/v1",
    ],
    "type": "VerifiablePresentation",
    "verifiableCredential": [{
      ...
      "proof": {
        "type": "RsaSignature2018",
        "created": "2017-06-18T21:19:10Z",
        "proofPurpose": "assertionMethod",
        "verificationMethod": "did:reclaim:1234#key-1",
        "proof": "abc"
      }
    }],
    "proof": {
      "type": "ReclaimPresentationSig2022",
      "created": "2022-2-1T0:0:0Z",
      "proofPurpose": "assertionMethod",
      "verificationMethod": "did:reclaim:ebfeb1f712...276e12ec21#keys-1",
      "signature": "2SWE7B8FWTNRG05...GWYX6SGQMKCC9B0"
    }
  }
}
```

The method then adds an empty signature field to the proof. The new JSON is then converted into a string without any whitespaces. The ReclaimPresentationSig2022 utilizes the cryptographic signature functions provided by GNUnet. We then create a package that contains metadata for the GNUnet signature algorithm and the message itself. *GNUNET_IDENTITY_sign_()* is executed. The returned signature is then encoded in base64 and added to the proof dictionary under the "signature" key. The JSON containing the proof is returned.

When verifying a presentation containing a ReclaimPresentationSig2022 signature the signing process is done backward. First, the signature is extracted from the JSON, decoded, and converted to a GNUnet signature. Second, the signature key-value pair is removed from the presentation proof. The presentation is then converted to a string as in the signature creation. Third, the signer's public key is extracted from the signer DID and converted to a GNUnet public key. The same GNUnet signature purpose is created. Forth, the *GNUNET_IDENTITY_signature_verify_()* is executed given the public key, purpose, and signature. The result of the GNUnet signature verification is returned.

The ReclaimPresentationSig2022 has disadvantages compared to other Linked Data proofs. It is not compatible with other DID implementations. Should the structure of the credential or presentation be changed two presentations containing the same information produce different hashes and therefore signatures.

4.4.4 Example

This section serves as an example for the ReclaimPresentationSig2022. Others can use it to verify their implementation of the ReclaimPresentationSig2022.

Given the Verifiable Credential:

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1"
  ],
  "type": [
    "VerifiableCredential"
  ],
  "issuer": "did:reclaim:1234",
  "issuanceDate": "2018-02-24T05:28:04Z",
  "expirationDate": "2025-02-24T00:00:00Z",
  "credentialSubject": {
```

```
    "id": "did:example:abcdef1234567",
    "name": "Tristan"
  },
  "proof": {
    "type": "RsaSignature2018",
    "created": "2017-06-18T21:19:10Z",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "did:reclaim:1234#key-1",
    "proof": "abc"
  }
}
```

and a GNUnet ego using EdDSA with private key:

```
000G053DJ5DV7RA35Y7BC8K7CW9H9HWKM3MFKD2W9NFSZ8TYS724E1KNN0
```

the implementations must create the following proof for the same created data:

```
"proof": {
  "type": "ReclaimPresentationSig2022",
  "created": "2022-02-12T20:15:46Z",
  "proofPurpose": "assertionMethod",
  "verificationMethod": "did:reclaim:000G057JMQ\
    3BHBZTN65AM83C32EX8EKCHASEJ33M4B1QNQEHMA6X2EFP74#key-1",
  "signature": "AAEAFBUU50t5kVbMEEQn0IDvhdEU+7ff\
    KuQAuuk1pRodysVFafFCwkD5bhOG3+k1Hu9RfGM+EGXFvnLu7Re4CAjRQgk="
}
```

Note: the backslash and linebreak in the *verificationMethod* and *signature* field is due to formatting and not present in the generated proof.

5 Discussion

In the following, Decentralized Identifiers will be discussed and then the difficulties with JSON-LD will be explained. In the end, the implementation of a DID method and Verifiable Credentials in re:claimID will be analyzed.

As described in 2.1.4, Mozilla criticized the Decentralized Identifier (DID) specification for several reasons. One of them is that DIDs use environmentally harmful distributed ledger technology and that they use centralized methods while claiming to be decentralized.

We think that these arguments are wrong. First, we argue that while the Bitcoin blockchain will most likely forever use Nakamoto consensus [20], and therefore depend on Proof-of-Work, other established blockchains (e.g. Ethereum) are moving to other consensus algorithms such as Proof-of-Stake. Proof-of-Stake promises to be more efficient than Nakamoto consensus and use significantly less energy while providing similar security [But17]. We think that the argument not to recommend the DID standard because DIDs can use blockchains is wrong since not all blockchains are environmentally harmful.

Centralized DID methods are allowed in the DID specification. Mozilla, therefore, argues that DIDs are not decentralized. However, the opposite is the case, the DID standard is decentralized in a way that it does not dictate what kind of technology to use. Everybody can build a DID method however he wants while still being interoperable. No central authority controls the development or usage of DID methods.

Our implementation of a DID method is neither centralized nor does it use distributed ledger technology. We present a new DID method that is a contradiction to the critique made by Mozilla.

Mozilla criticized [21c] the DID specification for encouraging divergence rather than convergence. We argue that the DID specification not defining the underlying technology used by DID methods and Verifiable Data Registry is a good thing. As of writing, at least 121 DID methods are being worked on [22e]. They range from fully centralized to fully decentralized and are being developed by different teams and independent developers. Probably all of them believe in their DID method and think that it has an advantage over other DID methods.

Them being interoperable and developers, in general, having the same goal of Self-Sovereign Identity, is the perfect situation for a competition to build the best DID methods. Saying this early in the development, that in the future hundreds of DID methods will coexist is highly speculative in our opinion. It could also be the case that only a few DID methods dominate the Self-Sovereign Identity space and supersede all others.

JSON-LD next to JSON is one of the data formats used by Verifiable Credentials/Presentations and Decentralized Identifier documents. JSON-LD still is very new and not supported in many programming languages (including C). We had trouble implementing Verifiable Credentials in re:claimID using JSON-LD, because we would need to implement not only a general JSON-LD library but also a JSON-LD proof compatible with other implementations. While our default DID document and the generated Verifiable Presentations are JSON-LD, our implementation does not use JSON-LD. An implementation must resolve and understand references in the *@context* list. The RDF Dataset Canonicalization [LS21] algorithm needed by JSON-LD proofs, is still work in progress. The concept of linked data is powerful but still new and adopting.

The methods a Verifiable Data Registry must provide map to the methods provided by the GNU Name System (create, replace, resolve, remove). Our implementation serves as a proof of concept that the new identity standard can be used with a decentralized identity directory. We see no obstacles to why the re:claimID DID method should not be further developed.

Verifiable Credentials also worked in re:claimID by making two shortcuts (no JSON-LD and no compatible proof). While multiple projects work on JSON-LD it would be easier to only focus on simple JSON Verifiable Credentials for now.

6 Future Work

6.1 Universal Resolver and Register

The Universal Register [Fou21b] and Universal Resolver [Fou21c] are projects from the Decentralized Identity Foundation. Using the Universal Register, a user can register/update/deactivate DID documents for multiple different DID methods. The Universal Resolver can be used to resolve any supported DID and supports a lot more (over 100) DID methods compared to the Universal Register (7). Both consist of Docker containers which all are accessible using the unified Swagger API. To add support for a DID method, a driver must be created which follows the requirements set by the Decentralized Identity Foundation. In future work, compatible drivers for the re:claimID DID method need to be created. While a driver for the Universal Resolver is doable with the existing proof-of-concept implementation, a driver for the Universal Register requires more work because it is unclear where keys are going to be stored. Docker containers for GNUnet including re:claimID already exist. The fastest way to serve a Swagger API might be to create an API with Python FastAPI [22f] and include that into the Docker Image.

6.2 re:claimID Decentralized Identifier Library

The current code base for DIDs and W3C Verifiable Credentials needs to be refactored, parts that fulfill different tasks need to be separated. GNUnet should be used as only a Verifiable Data Registry, without relying on GNUnet egos, re:claimID, and Verifiable Presentation generation. A user can store his own DID documents but relies on GNUnet to generate and manage his keys. This limits the implementation in a way that it is also responsible for the generation of Verifiable Presentations and implementing standard-compliant proofs while this is out of scope for a Verifiable Data Registry and done by a Self-Sovereign Identity (SSI) wallet. The next work should be to separate the functions for a Verifiable Data Registry from other re:claimID DID functions, making re:claimID and the Verifiable Data Registry provided by GNUnet independent from another.

6.3 Standard Compliant Linked Data Proof

The Proof currently used by the re:claimID Verifiable Credential implementation uses its own proof. GNUnet supports the digital signature algorithms, and digestAlgorithm used in other widely used Linked Data Proofs, e.g. "Ed25519 Signature 2018" [Sab21]. To standardize the re:claimID proof, a C-library for JSON-LD "RDF Dataset Canonicalization" is needed. A canonicalization C-library for RDF to make the proof compatible with other implementations seems like the natural next step. Using the canonicalization algorithm one could also implement more suffocated signature algorithms such as BBS+ signatures [21a] to support selective disclosure.

6.4 OpenID Connect

OpenID Connect supports self-issued OpenID Provider (Self-Issued OP). re:claimID provides a Self-Issued OP with its Browser extension. Connecting re:claimID with any website supporting Self-Issued OP. OpenID Connect also added support for W3C Verifiable Credentials [Ter+22]. In future work, one could extend the re:claimID Browser extension so it can display Verifiable Presentations created by the re:claimID W3C Verifiable Presentations plugin. For this, one would have to implement the OpenID Connect Presentation Extension [Ter+22]. This is only useful if the presentation proofs created by the plugin are compatible with other implementations.

6.5 DIDComm

DIDComm [Fou21a] is a communication protocol for DIDs. It supports communication over different (unsecured) channels while being secure, private, decentralized, interoperable, extensible, and efficient. GNUnet has its own communication application CADET [Man22] allowing secure end-to-end communication. In future work, one would create an extension for GNUnet CADET so DIDs can communicate with DIDComm using GNUnet as the underlying transport.

List of Figures

2.1 URIs, URLs, and URNs 4
2.2 Actors in a Verifiable Credential scheme 9
2.3 Actors and Components in a decentralized identity system 11
2.4 Zooko's triangle with GNU Name System and the Domain Name System 13

4.1 Creating and resolving a DID (document) 25
4.2 Removing a DID (document) 26
4.3 Dependencies between re:claimID, GNS, GNUnet Identity and plugins 30
4.4 Adding a Verifiable Credential to re:claimID and sharing a Verifiable Presentation 32

Bibliography

[00] *Efficient Proofs that a Committed Number Lies in an Interval*. 2000. URL: https://www.iacr.org/archive/eurocrypt2000/1807/18070437-new.pdf.

[01] *Names: Distributed, Secure, Human-Readable: Choose Two*. 2001. URL: https://web.archive.org/web/20011020191610/http://zooko.com/distnames.html.

[15] *JSON Web Token (JWT)*. 2015. URL: https://datatracker.ietf.org/doc/html/rfc7519.

[17a] *Edwards-Curve Digital Security Algorithm (EdDSA) for DNSSEC*. 2017. URL: https://datatracker.ietf.org/doc/html/rfc8080.html.

[17b] *Mastering Bitcoin*. O'Reilly Media, 2017. ISBN: 978-1491954386.

[17c] *The JavaScript Object Notation (JSON) Data Interchange Format*. 2017. URL: https://datatracker.ietf.org/doc/html/rfc8259.

[20] *Nakamoto Consensus*. 2020. URL: https://cs251.stanford.edu/lectures/lecture5.pdf.

[21a] *BBS+ Signatures 2020*. 2021. URL: https://w3c-ccg.github.io/ldp-bbs2020/.

[21b] *libpabc - Privacy-preserving Attribute-based Credentials*. 2021. URL: https://github.com/Fraunhofer-AISEC/libpabc.

[21c] *Response to 'Call for Review: Decentralized Identifiers (DIDs) v1.0 is a W3C Proposed Recommendation'*. 2021. URL: https://lists.w3.org/Archives/Public/public-new-work/2021Sep/0000.html.

[22a] *Credentials Community Group*. 2022. URL: https://www.w3.org/community/credentials/.

[22b] *Credentials Community Group*. 2022. URL: https://github.com/orgs/w3c-ccg/repositories.

[22c] *Decentralised Identity Foundation*. 2022. URL: https://identity.foundation/.

[22d] *Decentralised Identity Foundation Projects*. 2022. URL: https://github.com/orgs/decentralized-identity/repositories.

[22e] *DID Specification Registries*. 2022. URL: https://w3c.github.io/did-spec-registries/#did-methods.

[22f] *FastAPI*. 2022. URL: https://github.com/tiangolo/fastapi.

[22g] *GNUnet - The Internet of tomorrow needs GNUnet today*. 2022. URL: https://www.gnunet.org/en/.

[22h] *GNUnet DID Implementation*. 2022. URL: https://git.gnunet.org/gnunet.git/log/?h=dev/trizuz/dids.

[22i] *GNUnet Reference Manual*. 2022. URL: https://docs.gnunet.org/handbook/gnunet.html.

[22j] *GNUnet Verifibale Credentials Implementation*. 2022. URL: https://git.gnunet.org/gnunet.git/log/?h=dev/trizuz/w3cvc.

[22k] *Hyperledger Aries Projects*. 2022. URL: https://github.com/hyperledger/aries.

[22l] *OpenID Foundation*. 2022. URL: https://openid.net/foundation/.

[22m] *re:claimID - Self-sovereign, Decentralised Identity Management and Personal Data Sharing*. 2022. URL: https://www.gnunet.org/en/reclaim/index.html.

[22n] *Rebooting Web-of-Trust*. 2022. URL: https://www.weboftrust.info/index.html.

[22o] *The BBS Signature Scheme*. 2022. URL: https://identity.foundation/bbs-signature/draft-bbs-signatures.html.

[22p] *The GNU Name System*. 2022. URL: https://lsd.gnunet.org/lsd0001/.

[22q] *W3C Mission*. 2022. URL: https://www.w3.org/Consortium/mission#vision.

[22r] *Whitepaper from Rebooting Web-of-Trust*. 2022. URL: https://www.weboftrust.info/papers.html.

[87] *Domain Names - Implementation and Specification*. 1987. URL: https://datatracker.ietf.org/doc/html/rfc1035.

[All+19] C. Allen, K. H. Duffy, R. Grant, and D. Pape. *BTCR DID Method*. 2019. URL: https://w3c-ccg.github.io/didm-btcr/.

[BFM05] T. Berners-Lee, R. Fielding, and L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. 2005. URL: https://www.rfc-editor.org/rfc/rfc3986.html.

[Bro10] D. R. L. Brown. *SEC 2: Recommended Elliptic Curve Domain Parameters*. 2010. URL: https://www.secg.org/sec2-v2.pdf.

[BS22] J. Benet and M. Sporny. *The Multibase Data Format*. 2022. URL: https://www.ietf.org/id/draft-multiformats-multibase-04.html.

[But17] V. Buterin. *Proof of Stake FAQ*. 2017. URL: https://vitalik.ca/general/2017/12/31/pos_faq.html.

[Coo+13] A. Cooper, H. Tschofenig, B. Aboba, J. Peterson, J. Morris, M. Hansen, and R. Smith. *Privacy Considerations for Internet Protocols*. 2013. URL: https://www.rfc-editor.org/rfc/rfc6973.

[FIS05] FIST. *ANSI X9.62 - Public Key Cryptography for the Financial Services Industry: the Elliptic Curve Digital Signature Algorithm (ECDSA)*. 2005.

[Fou21a] D. I. Foundation. *DIDComm*. 2021. URL: https://didcomm.org/.

[Fou21b] D. I. Foundation. *Universal Registrar*. 2021. URL: https://github.com/decentralized-identity/universal-registrar/.

[Fou21c] D. I. Foundation. *Universal Resolver*. 2021. URL: https://github.com/decentralized-identity/universal-resolver/.

[Gri+21] C. Gribneau, M. Prorock, O. Steele, O. Terbu, M. Xu, and D. Zagidulin. *did:web Method Specification*. 2021. URL: https://w3c-ccg.github.io/did-method-web/.

[HB21] D. Huseby and K. Bull. *Hyperledger Aries*. 2021. URL: https://wiki.hyperledger.org/display/ARIES/Hyperledger+Aries.

[LS21] D. Longley and M. Sporny. *RDF Dataset Canonicalization*. 2021. URL: https://json-ld.github.io/rdf-dataset-canonicalization/spec/.

[Man22] G. R. Manual. *CADET Subsystem*. 2022. URL: https://docs.gnunet.org/handbook/gnunet.html#CADET-Subsystem.

[RK03] E. Rescorla and B. Korver. *Guidelines for Writing RFC Text on Security Considerations*. 2003. URL: https://www.rfc-editor.org/rfc/rfc3552.html.

[Sab21] M. Sabadello. *Ed25519 Signature 2018*. 2021. URL: https://w3c-ccg.github.io/lds-ed25519-2018/.

[SLC21a] M. Sporny, D. Longley, and D. Chadwick. *Verifiable Credentials Data Model*. 2021. URL: https://www.w3.org/TR/vc-data-model/.

[SLC21b] M. Sporny, D. Longley, and D. Chadwick. *Verifiable Credentials Data Model - Zero Knowledge Proofs*. 2021. URL: https://www.w.org/TR/vc-data-model/#zero-knowledge-proofs.

[Spo+20] M. Sporny, D. Longley, G. Kellogg, M. Lanthaler, P.-A. Champin, and N. Lindström. *JSON-LD 1.1 - A JSON-based Serialization for Linked Data*. 2020. URL: https://www.w3.org/TR/json-ld/.

[Spo+21] M. Sporny, D. Longley, M. Sabadello, D. Reed, O. Steele, and C. Allen. *Decentralized Identifiers (DIDs) v1.0*. 2021. URL: https://w3c.github.io/did-core/.

[Ter+22] O. Terbu, T. Lodderstedt, K. Yasuda, A. Lemmon, and T. Looker. *OpenID Connect for Verifiable Presentations*. 2022. URL: https://openid.net/specs/openid-connect-4-verifiable-presentations-1_0.html.

[W3C20] C. C. G. (W3C). *Linked Data Cryptographic Suite Registry*. 2020. URL: https://w3c-ccg.github.io/ld-cryptosuite-registry/.