



Driving innovation together

A thin orange line starts horizontally from the end of the "Driving innovation together" text, then curves downwards and to the right, ending horizontally again.

Technical exploration Ledger-based Self Sovereign Identity

Table of contents

Management summary	4
1 Introduction	5
2 What is Self Sovereign Identity?	7
2.1 Introduction	7
2.2 Anonymity and Self-Control	8
2.3 A decentralised identity ecosystem	9
2.4 A layered architecture	10
2.5 Technology stack	11
2.6 Trust framework	17
3 Project activities	19
3.1 Research proposal	19
3.2 Architecture and technology deep dive	19
3.3 Search for solutions	20
3.4 Assessing suitability	20
3.5 Platform selection.....	21
3.6 Setting up the infrastructure	22
3.7 Building out the experiment.....	23
3.8 Define use cases	23
3.9 Documentation of findings	24
4 Introducing the BCGov stack	25
4.1 Ledger	25
4.2 The OrgBook	26
4.3 Greenlight demos	26
4.4 Messaging ecosystem.....	27
5 Implementation of a SURF SSI demo	28
5.1 Ledger setup	28
5.2 PHP-based Issuer/Verifier.....	28
5.3 Setup using ACA-Py.....	28
5.4 Setup using Django/React	29
5.5 Setup using ACA-Py with Django/React.....	29



5.6	Information flows	30
5.7	Deployment	34
6	Findings.....	38
6.1	Use cases	38
6.2	Maturity	38
6.3	Interoperability	39
6.4	Scalability	40
6.5	Privacy.....	41
6.6	Revocation	41
6.7	Trust framework	43
7	Follow-up	49
8	Conclusions	51
9	Online resources.....	54

Management summary

Self Sovereign Identity (SSI) introduces a new paradigm in which users become direct controllers of their own personal data. Profile information is collected by users from multiple authoritative sources (issuers) into a single digital 'wallet'. Upon request, users may release this information selectively directly from their wallet to a service. Because of this, the SSI model is very appealing from the perspective of privacy and data protection. In the SSI model, issuance of personal data is decoupled from issuers and receiving services. This reduces the number of entities processing personal data and prevents both issuers and services from inferring additional personal details about users through transaction metadata.

Authentication and Authorisation Infrastructures (AAI) support many of the core processes in Research and Education (R&E). Currently, the AAI model builds on federated identity, where user identities are managed by their institution. Federated identity is also common outside R&E, e.g., with social network providers or government issued identity. However, with federated identity, the identity provider determines the use of the identity and hence, by definition, also limits it. This leads to siloes and prevents re-use of already available profile information. This may incur unnecessary costs as identity and profile information needs to be re-established by services which are not able to use the user's identity provider and also leads to more hassle for end users. The only scalable way to combine profile information from multiple autonomous sources typically introduces a proxy service which brings various new challenges around availability, security and data protection. By effectively making the end user the 'proxy' who aggregates all personal data, the SSI model may potentially help to avoid many of these issues.

However, the SSI model does introduce several challenges, the most important of which is building trust between recipients and issuers. To achieve this, SSI introduces the concept of a verifiable data registry, typically implemented using a distributed ledger or blockchain. It holds an immutable registry of all transactions (but not the personal data itself). In the activity described in this report we investigated the (technical) characteristics, standards and the implementation of a SSI solution using a verifiable data registry based on a blockchain. We also assessed the maturity and usability of a blockchain-based solution by deploying and testing it with other components in the AAI ecosystem, in accordance to use cases we have collected.

As a general concept, the privacy-preserving nature of SSI, end user control over the disclosure of personal data and SSI's trust model aligned well with the public values typically found in R&E. The platform we used (based on Hyperledger Indy) allowed us to perform all use cases successfully. The platform lives up to the promises of SSI: being privacy preserving, scalable and secure. The user interaction and user interfaces are however the weakest part of the ecosystem and will need a lot of attention.

Other sectors, including government and healthcare, are increasingly investigating SSI. For SURF, several services could potentially benefit from SSI, which merits further investigation. SURF should also consider its role in a future SSI ecosystem, either within or outside the R&E sector.

1 Introduction

Authentication and authorisation support many of the core processes in Research and Education (R&E). Campus applications, whether on-premise or in the cloud, as well as (distributed) research services, must have the ability to determine a user's identity to enable safe and secure access to content and resources. Typically, an authentication procedure is made up of two elements. Authentication is where a user proves that they can present certain credentials. If successful, authentication is followed by the release of attributes that reveal a set or subset of profile information about the user to the service. The combined result is evaluated by the service, allowing the service to determine whether the user should be authorised to access the service and the associated content.

In the early days, access to resources was managed on the basis of local accounts, where credentials and the profile of a user reside on the machine hosting the service. As the service holds all relevant aspects of the user's identity locally, this model scales poorly over multiple services and also forces the user to memorise many sets of credentials.

Over time, the R&E sector has adopted the federated access model. In this model, the users' credentials and profile information are managed by the home institution. By merit of technical standardisation, trust federations and contracts, the user is able to authenticate at services based on a login profile with their institutional identity provider. The major benefit of this model is that the service does not need to hold a users' credentials, and the institutional account can be used for access to multiple services. Outside of R&E, the use of federated access is very common, as many social network providers are more than happy to act as an identity provider.

While the federated model has many advantages over the previous one, the role of the identity provider comes with challenges, especially because it is the identity provider that determines the use of the identity and hence, by definition, also limits it.

Self Sovereign Identity (SSI) introduces a new paradigm where the user becomes the direct controller of all profile information. The information is collected by the user from multiple authoritative sources into a single 'wallet'. On request, the user may release this information selectively to a service. Within this ecosystem, the act of authentication is separated from the delivery of profile information, called 'claims'. And even better, the delivery of attributes does not have to be performed in real time. The SSI model is very appealing from the perspective of privacy and data protection, as the user is always involved in the release of personal data and has direct control over this process. In addition, all interactions that deal with data have the wallet as an intermediary. As a result, neither the authoritative issuer of the data nor the recipients, can profile a user's behaviour.

However, the SSI model does introduce several challenges, the most important of which is building trust between the recipient and the issuers. How does the verifier know the data presented by the user is legitimate and issued by a trusted party? To achieve this, SSI introduces the concept of verifiable data through the verifiable data registry. This registry is typically implemented using a distributed ledger or blockchain. It holds an immutable registry of all transactions (but not the transaction data itself) made by various parties in the ecosystem. Additional transparency may be provided by making the registry public.

As a general concept, the privacy-preserving nature of SSI, end user control over the disclosure of attributes and also SSI's trust model are well aligned with the public values typically found in academia. In fact, very similar considerations have in part at least led to the principles by which R&E federations are operated and governed, such as data minimisation and the distributed mesh model commonly used in today's identity federations.



SSI is a novel concept and is still in development both in terms of standards and the availability of technical implementations. Novel or not, SSI has managed to attract a lot of attention in recent years, not only from Silicon Valley-based start-ups, but also from large tech companies, healthcare providers and government agencies globally.

The study described in this document was carried out between September and December 2020. At SURF, we have previously gained experience with the use of IRMA (“I Reveal My Attributes”¹), a distributed identity platform that does not use a distributed ledger as its verifiable data registry. In this study, we wanted to investigate the state of distributed ledger-based solutions from a technical perspective and from a functional perspective. We have identified potential use cases in the context of SURF and investigated which technical SSI implementations might fit these. We then selected one of the platforms and deployed this to learn about its features, capabilities but also challenges. We concluded by testing the various use cases with this platform.

This report begins by describing SSI and its concepts, standards and components in more detail in chapter two. Chapter three describes the project activities that were conducted. In chapter four, we describe the technology stack we selected in more detail, to set the stage for chapter five, where we describe the SURF deployment we set up for further testing. In chapter six, we present the findings, both when evaluating the standards and platforms, our experiments with the technical setup we deployed and while testing our use cases. Unfortunately, our time was limited and, at the same, time more insight also brought additional questions and ideas. Chapter seven therefore contains a number of suggestions for further work that could be conducted.

Chapter eight wraps up the document with a number of conclusions on SSI and its applicability in the context of SURF. We have also provided links to online resources including the git repositories, where we have made all our code and deployment information available, as well as some demonstration videos.

¹ <https://privacybydesign.foundation/en/>

2 What is Self Sovereign Identity?

2.1 Introduction

Self Sovereign Identity is a concept that deals with the way a user's identity is managed and used in the digital world. Users must be able to create and control their own identity without relying on a centralised authority.

There are many definitions of Self Sovereign Identity (SSI) in circulation. Although there is no consensus on an exact definition of Self Sovereign identity yet, there are common grounds. The Sovrin project offers a good explanation of the phenomenon and of the philosophy of SSI².

“Self Sovereign identity is a term used to describe the digital movement that recognises an individual should own and control their identity without the intervening administrative authorities. SSI allows people to interact in the digital world with the same freedom and capacity for trust as they do in the offline world.

Everyone (including businesses and IoT) has different relationships or unique sets of identifying information. This information could be things like birth date, citizenship, university degrees, or business licenses. In the physical world, these are represented as cards and certificates that are held by the identity holder in their wallet or safe place like a safety deposit box, and are presented when the person needs to prove their identity or something about their identity.

Self Sovereign identity brings the same freedoms and personal autonomy to the internet in a safe and trustworthy system of identity management. SSI means the individual (or organisation) manages the elements that make up their identity and controls access to those credentials – digitally. With SSI, the power to control personal data resides with the individual, and not an administrative third party granting or tracking access to these credentials.

The SSI identity system gives you the ability to use your digital wallet and authenticate your own identity using the credentials you have been issued. You no longer have to give up control of personal information to dozens of databases each time you want to access new goods and services, with the risk of your identity being stolen by hackers.

This is called “Self Sovereign” identity because each person is now in control of their own identity—they are their own sovereign nation. People can control their own information and relationships. A person’s digital existence is now independent of any organisation: no-one can take their identity away.”

² <https://sovrin.org/faq/what-is-Self-Sovereign-identity/>

Ten principles of Self Sovereign Identity

Ten guiding principles were deemed essential to the concept of SSI³. These principles help to define the architecture of SSI. They are successively:

1. Existence: users must have an independent existence.
2. Control: users must control their identities.
3. Access: users must have access to their own data.
4. Transparency: systems and algorithms must be transparent.
5. Persistence: identities must be long-lived.
6. Portability: information and services about identity must be transportable.
7. Interoperability: identities should be as widely usable as possible.
8. Consent: users must agree to the use of their identity.
9. Minimisation: disclosure of claims must be minimised.
10. Protection: the rights of users must be protected.

In conclusion, SSI is actually about two aspects: on the one hand, SSI an ideology that a user owns his or her own identity and can therefore interact in the digital world with the same freedom and capacity for trust as they do in the offline world. On the other hand, SSI is all about the architecture and associated technology to make this ideology possible.

2.2 Anonymity and Self-Control

Using the above principles of SSI, it becomes clear that maintaining anonymity is of great importance. If the protocols used leak information about a user, that user is no longer in control of his identity (principle 2) or able to give consent about sharing their identity (principle 8).

For this reason, the following supporting principles should serve as a guide for any implementation:^{4 5}

1. Data minimisation: never share more than the user intends to share.
2. Unlinkability: colluding parties should not be able to determine more information about the user by linking information contained in the activities (requesting attributes, presenting proof).
3. Pseudonymity: each interaction with the user can be delegated to a pseudonym that uniquely defines the interaction and is not used outside this context.
4. Unobservability: observing activities should not leak additional information about the user or users performing the activities.

To indicate the abundant use of cryptography to ensure anonymity and unlinkability, the terms are referenced at relevant points in this document.

Not all SSI implementations are concerned about these supporting principles to the same degree. Some solutions, for example, never anonymise users, while others use a central registration system that can easily link activity to users. Many implementations do not just leverage the technical infrastructure for establishing trust, but add a non-technical policy layer too.

³ <https://github.com/WebOfTrustInfo/Self-Sovereign-identity/blob/master/ThePathToSelf-SovereignIdentity.md>

⁴ <https://tools.ietf.org/id/draft-hansen-privacy-terminology-00.html>

⁵ https://dud.inf.tu-dresden.de/Anon_Terminology.shtml

2.3 A decentralised identity ecosystem

The concept of creating a unified, decentralised identity ecosystem tries to support a set of fundamental user needs, yet at the same time must overcome several technical and organisational challenges, some of which are very novel to the identity space⁶:

- Enabling registration of Self Sovereign identifiers which no provider owns or controls.
- The ability to issue, look up, discover and verify identifiers and claims across distributed decentralised systems, where there is potentially no centralised ownership or trust
- Providing an understandable and user-friendly mechanism for users (or 'subjects' as they are called in SSI) to securely store sensitive personal identity data and enabling subjects to precisely control what is shared with others.

Self Sovereign Identity is based on the use of Decentralised Identifiers (DID⁷). As we will learn later on in the technical chapter, a DID is just an identifier; it does not provide information about the Subject itself. In practice, DIDs are used in combination with Verifiable Claims (VC⁸) to support digital interactions in which information about the Subject must be shared with third parties, by proving to those third parties that the DID Subject owns a certain set of attestations or claims. This proof is based on the cryptographic link between the VC, the DID Subject the VC relates to, and the Issuer of the VC, which can be the DID Subject (self-asserted claims), or a trusted entity. In other words, the VCs are signed with the private key associated with the public key associated with the issuer's DID.

Trust of the Issuer is established either by trusting the Issuer's DID (e.g. out-of-band, bilateral relationship, trusted lists) or by any other means. The third party can then use the cryptographically protected proof presented to verify the ownership and trustworthiness of the claims about the Subject. As the presentation of the claims is managed by the users, they can decide which specific pieces of information about themselves they want to share with third parties. This selective disclosure of attributes is believed to reinforce privacy and personal data protection.

The flow of information between entities involved in the verifiable claims generation and its subsequent use are depicted in Figure 1 below, taken from the W3C working draft of the Verifiable Credentials Data Model (1.0)⁹. In this Data Model, credentials are considered to be a set of one or more claims made by an Issuer. When implementing DID and VC for use in Self Sovereign Identity scenarios, entities rely on the use of a Distributed Ledger or Blockchain technology to support the Verifiable Data Registry DIDs.

⁶ <https://medium.com/decentralized-identity/the-rising-tide-of-decentralized-identity-2e163e4ec663>

⁷ <https://www.w3.org/TR/did-core/>

⁸ <https://www.w3.org/TR/vc-data-model/>

⁹ <https://w3c-ccg.github.io/did-spec/>

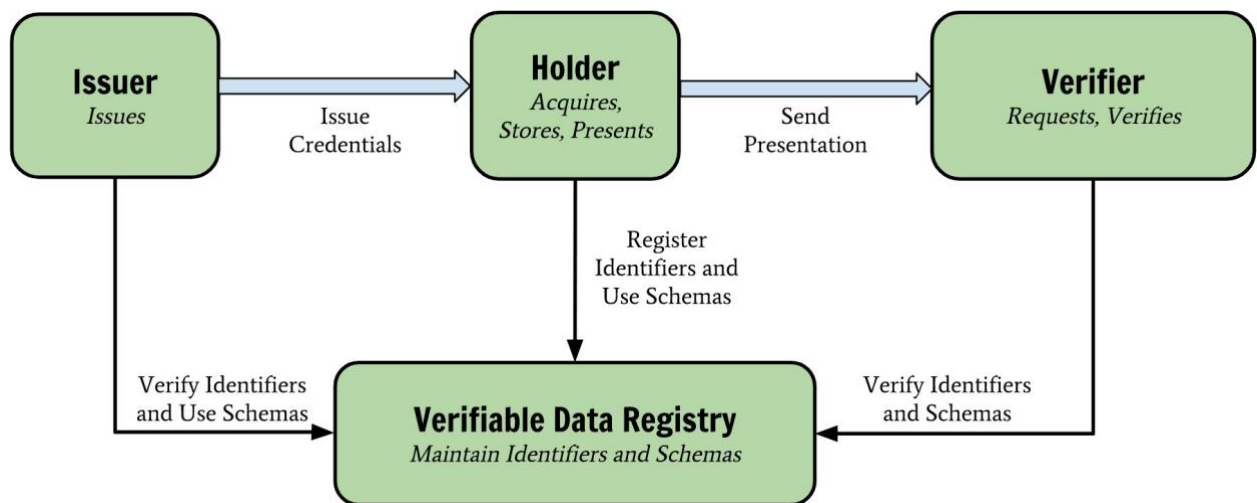


Figure 1: Information flow in use of Verifiable Credentials¹⁰

2.4 A layered architecture

The SSI ecosystem combines a number of existing and emerging concepts and technologies. The Decentralised Identity Foundation (DIF¹¹) is proposing an architecture based on the following components:

- User agent: a program, such as a browser, mobile app or other web client, that mediates the communication between holders, Issuers, and Verifiers;
- Identity Hubs: secure personal data stores that coordinate storage of signed/encrypted data, and relay messages to identity-linked devices;
- Universal Resolver: a server featuring a pluggable system of DID Method drivers that enables resolution and discovery of DIDs across any decentralised system;
- Universal Registrar: a server that enables the registration of DIDs across any decentralised system that produces a compatible driver.

This layered architecture can be seen with the Sovrin project, as shown in Figure 2 below.

¹⁰ <https://www.w3.org/TR/vc-data-model/>

¹¹ <https://identity.foundation/>

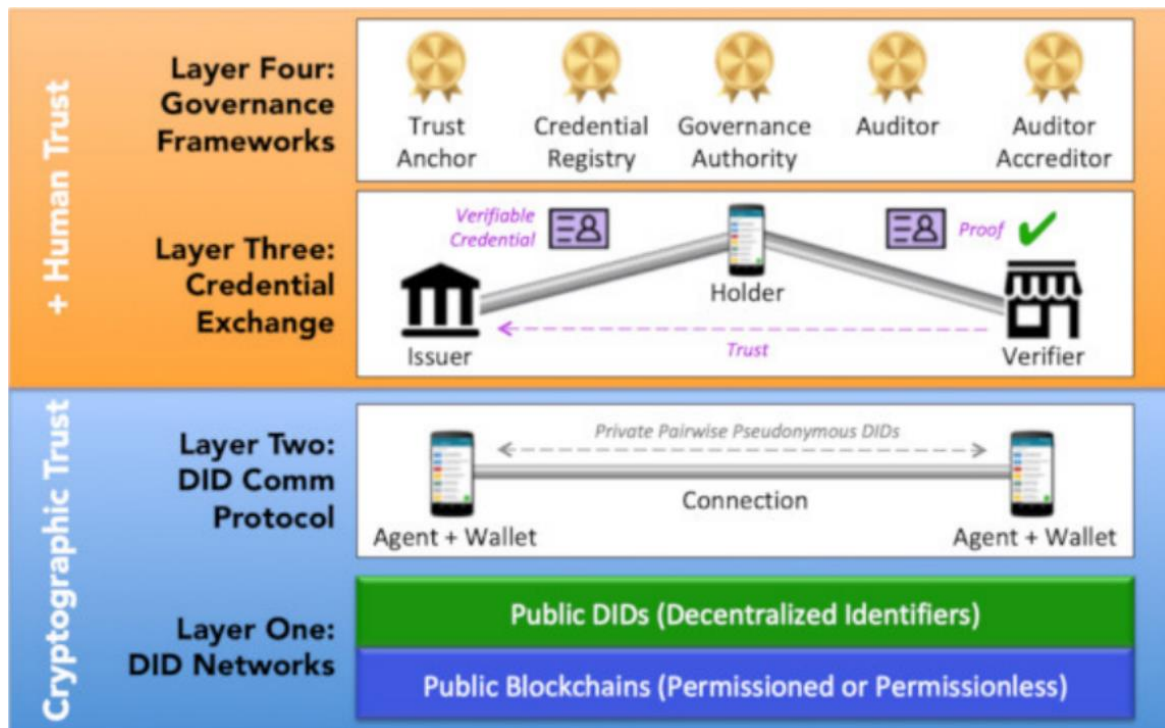


Figure 2: SSI components as described by the Decentralised Identity Foundation in case of the Sovrin network¹²

In Layer One, a Public Blockchain acts as the Universal Registrar component, whereas the Public DIDs are the implementation of the Universal Resolver component. In Layer Two, the DIF components Identity Hub is depicted as the wallet and the DIF User agent component is shown as Agent. In addition to these DIF-components, Sovrin adds the Credential Exchange in Layer Three. It adds the actual exchange of Verifiable Claims between the Issuer, Holder and Verifier.

On top of all this, a governance framework is added, as shown in Layer Four. This layer is similar to existing federation policies.

2.5 Technology stack

The SSI functionality is implemented based on four major open standards; this chapter will discuss the properties and challenges associated with the various standards and protocols:

- DID,
- DKMS,
- DIDauth and
- Verifiable Credentials

Figure 3 shows where in the ecosystem the various standards come into play. While some standards are already established and mature, others are still under heavy development.

¹² <https://www.evernym.com/blog/hyperledger-aries/>

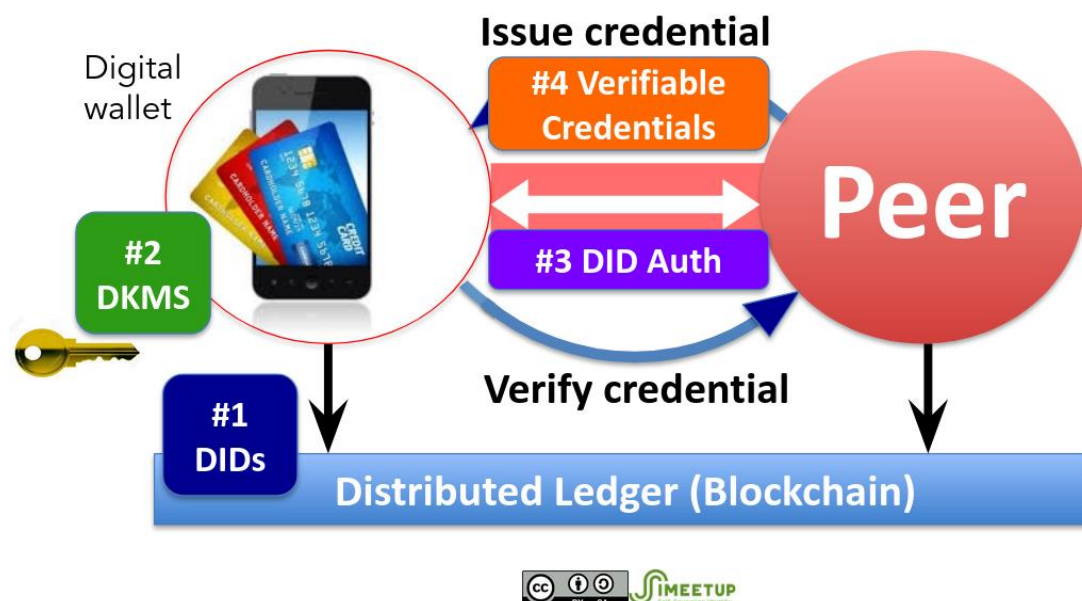


Figure 3: standards used in the SSI ecosystem¹³

A very recent open standard that could become the de facto standard SSI technology stack is DIDComm Messaging¹⁴. It is a powerful way for people, institutions, and IoT devices to interact via machine-readable messages, using features of DIDs as the basis of security and privacy. It works over any transport: HTTP, Bluetooth, SMTP or raw sockets, for example. Currently, specifications are being developed as reference code to explain DIDComm Messaging. It is likely to be standardised at IETF or other fora. Some of its implementations (e.g., Sovrin) use early versions of DIDComm.

2.5.1 DID

Decentralised identifiers (DIDs) are a new type of identifier that enable verifiable, decentralised digital identity. A DID identifies any Subject (e.g., a person, organisation, thing, data model, abstract entity, etc.) that the controller of the DID decides that it wants to identify. DID is a W3C standard¹⁵.

“Decentralised Identifiers (DIDs) are a new type of identifier for verifiable, “Self Sovereign” digital identity. DIDs are fully under the control of the DID Subject, independent from any centralised registry, identity provider or certificate authority. DIDs are URLs that relate a DID Subject to means for trustable interactions with that Subject. DIDs resolve to DID Documents — simple documents that describe how to use that specific DID. Each DID Document must contain at least three things: proof purposes, verification methods, and service endpoints. Proof purposes are combined with verification methods to provide mechanisms for proving things. For example, a DID Document can specify that a particular verification method, such as a cryptographic public key or pseudonymous biometric protocol, can be used to verify a proof that was created for the purpose of authentication. Service endpoints enable trusted interactions with the DID controller”

In contrast to typical federated identifiers, DIDs have been designed so that they may be decoupled from centralised registries, identity providers, and certificate authorities. Specifically, while other parties might be used to help enable the discovery of information related to a DID, the design enables the controller of a DID to prove control over it *without* requiring permission from any other party. DIDs are URIs that associate a DID

¹³ <https://ssimeetup.org/decentralized-identifiers-did-fundamental-block-Self-Sovereign-identity-drummond-reed-webinar-2/>

¹⁴ <https://github.com/decentralized-identity/didcomm-messaging>

¹⁵ <https://www.w3.org/TR/did-core/>

Subject with a DID document, allowing trustable interactions associated with that Subject.

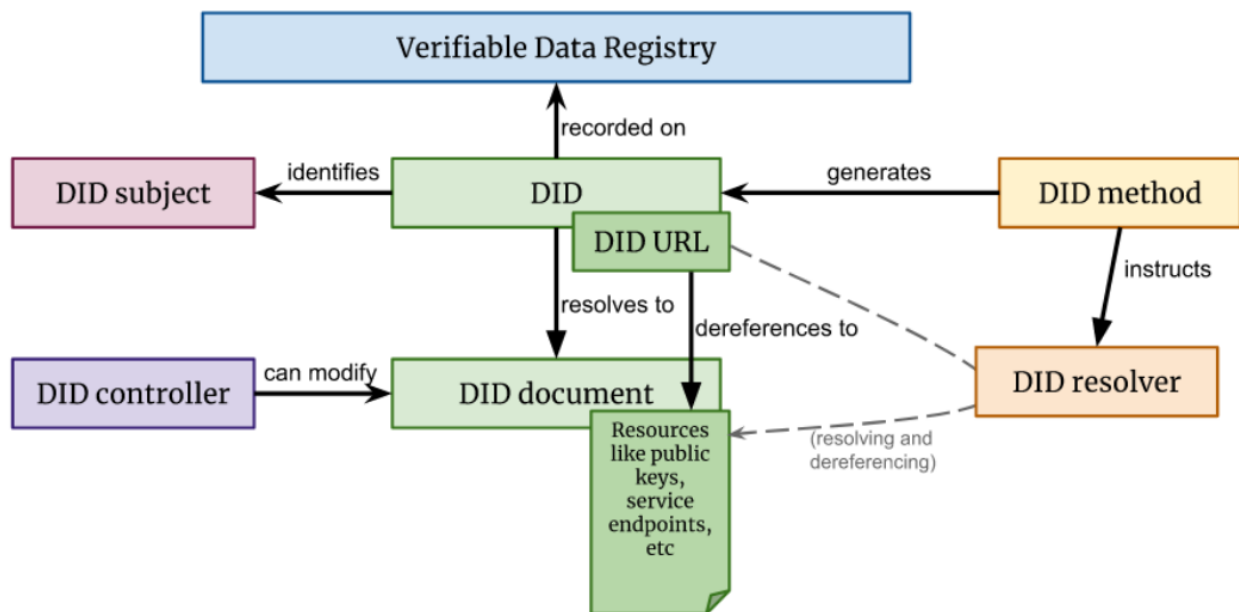


Figure 4: The basic components of DID architecture ¹⁶

DID Method

The DID method specification must specify how a client creates a DID record—the combination of a DID and its associated document—on the target system, including all cryptographic operations necessary to establish proof of ownership.

All entities which use the same DID method often run the same “network”, e.g. the Sovrin network which uses the “did:sovrin” method. Given that the DID method defines in technical terms how a DID may be issued and verified, it is a core element of the trust framework.

DID Document

DID documents contain metadata associated with a DID. They typically express verification methods (such as public keys) and services relevant to interactions with the DID Subject. A DID document represents an abstract data model, and can be serialised according to a particular syntax. Commonly, JSON-LD (JSON for Linking Data) is used to describe relationships. A DID Document itself does not have to be saved on the ledger, as you can also point to it. Since a DID Document is merely an address, you need to be able to trust the publisher of this address. This publisher states that the addressed DID Document is under its control.

2.5.2 DKMS

DKMS, an abbreviation for Decentralised Key Management System, is an open standard for managing DIDs and private keys of users. DKMS is a new approach to cryptographic key management intended for use with blockchain and distributed ledger technologies where there are no centralised authorities. DKMS inverts a core assumption of conventional PKI (public key infrastructure) architecture, namely that public key certificates will be issued by centralised or federated certificate authorities. With DKMS, the initial “root of trust” for all participants is any distributed ledger that supports a new form of root identity record called a DID

¹⁶ <https://www.w3.org/TR/did-core/#architecture-overview>

(decentralised identifier). The DKMS draft¹⁷ is in preparation to be submitted to a standards development organisation such as OASIS for formal standardisation.

In the SSI ecosystem, DKMS applies to the wallets where you store your DIDs and private keys and to the agents that read/write from those wallets.

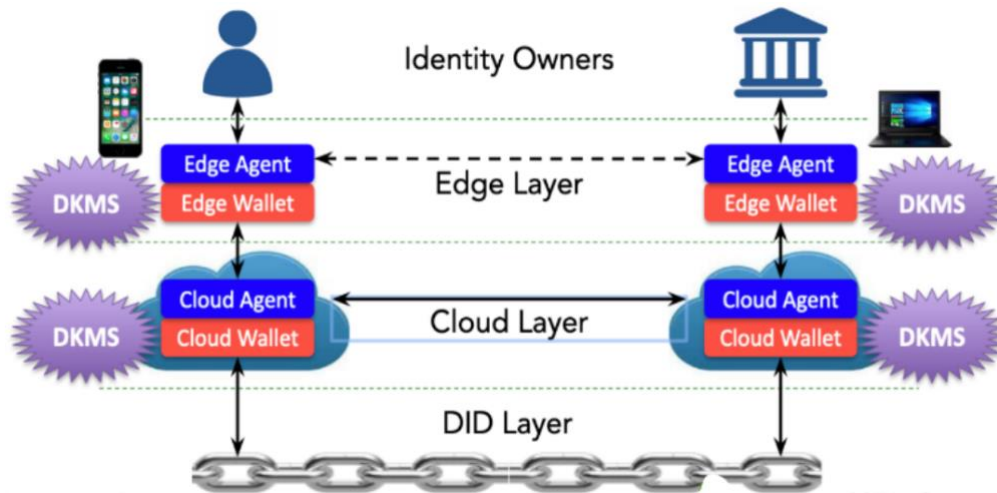


Figure 5: DKMS in decentralised identity stack ¹⁸

Given that DKMS relies on public/private key pairs, two aspects are critical to the secure and trustworthy operational use of these keys: key recovery and key rotation.

Key Recovery

The ability to recover keys in case of loss or theft is critical. DKMS key recovery supports both offline recovery (“paper wallet”) and social recovery (“trustee”) methods. Both are based on cloud agents continuously storing a backup copy of your wallet encrypted with a special recovery key.

Key Rotation

In the event that an attacker compromises a Holder’s private key, the keys associated with that identity must be replaced with a non-compromised pair in order for the Holder to regain (exclusive) control over the linked identity. A process of key rotation is used to re-secure the Holder’s identity. Multiple methods for performing key rotation are available using DID Documents¹⁹.

It is advised to use different keys for different purposes.

- Master Key
- Key Encryption keys

¹⁷ <https://github.com/hyperledger/aries-rfcs/blob/master/concepts/0051-dkms/dkms-v4.md>

¹⁸ <https://ssimeetup.org/decentralized-key-management-dkms-essential-missing-piece-ssi-puzzle-drummond-reed-webinar-8/>

¹⁹ <https://jolocom.io/wp-content/uploads/2019/12/Jolocom-Whitepaper-v2.1-A-Decentralized-Open-Source-Solution-for-Digital-Identity-and-Access-Management.pdf>

- Data keys

The keys are listed from the highest to lowest order. The lower keys can be re-created / rotated from the higher ones. The Master key, the highest, should be securely stored offline.

If the Master key is compromised and abused for key rotation of an attacker, there is no recovery from that.

Hyperledger supports a concept of 'Agent Policies', where the ledger documents delegate access to certain DIDs for certain agents (wallet, Cloud Agent, etc.). This concept is currently not implemented, but would allow users to selectively disable an Agent after it has been compromised, or to recover from a compromise situation by using a combination of uncompromised agents to recover a key. The main suggestions for this recovery situation revolve around sharding.

It is also possible to perform a key rollover for a given DID on the ledger. In theory, the last transaction for a DID defines the active public key used for that DID. By creating a new transaction, the former public key can be invalidated. This transaction has to be signed with the former public key though, causing problems where that key has been lost or if it was compromised and immediately used to roll over the key. However, as transactions need to be signed by a limited and well-known list of trust anchors, the culprit in case of a stolen-and-replaced key would be easy to find.

DKMS architecture and DPKI provide the following major benefits:

- No single point of failure: there is no central CA or other registration authority;
- Interoperability – open standard: perform key exchange and create encrypted P2P connections without reliance on proprietary software, service providers or federations;
- Resilient trust infrastructure: decentralised access to cryptographically verifiable data;
- Key recovery: robust key recovery directly into the infrastructure.

Key Revocation

From our research, we conclude that key revocation is not part of the SSI standards in any form. This means that future functionality concerning key revocation will be offered by platforms outside the scope of the standard.

2.5.3 DIDauth

The DIDauth standard is used to prove that the user is in control of a DID. DIDAuth was invented at the 'Rebooting the Web of Trust'²⁰ effort, following the observation that a number of initiatives had already devised their own authentication solutions without an intermediate IdP.

A DID contains one or more public keys or other one-way results of functions (such as biometric-based templates). The user can demonstrate that they are in possession of information that can be verified by the public key (for example, a private key signature).

In addition to the keys, a DID can also define one or more authorisation service endpoints to which relying parties can send a challenge to check possession of the private key associated with the public key of the DID. It is assumed that those authorisation service endpoints are under the control of the user and notify the user of the requested challenges. Specifically, biometric authorisation is taken into account. Authorisation service

²⁰ <https://www.weboftrust.info/>

endpoints forward challenges to a wallet implementation, for example, where the user can solve (decrypt and prove) the challenge using the securely stored private key.

Replay and man-in-the-middle attacks are, of course, applicable here. The protocol is broad and vague, and includes some specific implementations or variations of existing protocols. There does not appear to be an active implementation of any version of the suggested DIDAuth protocols. This leads to the conclusion that DIDAuth itself is more a concept than an actual protocol.

It appears that DIDAuth, as described in 2016, has been supplanted by the W3C specification for DIDcomm, although not all features as envisioned for DIDAuth are available in DIDcomm and vice versa. DIDcomm is concerned with routing cryptographically secure messages through a landscape of endpoints in such a way that sender and receiver can be assured of secure and private communication. This includes creating a connection between sender and receiver, which is similar to the initial identification/authentication step described in DIDAuth.

2.5.4 Verifiable Credentials

A verifiable credential (VC) is a tamper-evident credential that has authorship that can be cryptographically verified. A verifiable credential can represent all of the same information that a physical credential represents. The addition of technologies, such as digital signatures, makes verifiable credentials more tamper-evident and more trustworthy than their physical counterparts. A verifiable claim is a qualification, achievement, quality, or piece of information about an entity's background. Examples include a name, government ID, payment provider, home address or university degree. Verifiable credentials is a W3C standard²¹.

A VC can contain the evidence field as included by an Issuer to provide the Verifier with additional supporting information on a verifiable credential. This could be used by the Verifier to establish the confidence with which it relies on the claims in the verifiable credential.

Verifiable Presentation

A verifiable presentation expresses data from one or more verifiable credentials and is packaged in such a way that the authorship of the data is verifiable. It becomes a tamper-evident presentation encoded in such a way that authorship of the data can be trusted after a process of cryptographic verification. A verifiable presentation is requested by a Verifier and created by a Holder.

Verifiers can request specific attributes from a potentially limited list of trusted credential definitions or schemas, as published on the ledger, or request proof of non-revocation within a certain timeframe. It is also possible to request proof of a so-called predicate on an attribute. In that case, the attribute is converted to a number and the verifier can request 'less than' or 'larger than' operations on the number. Obvious use-cases for these numbers are date of birth and income.

²¹ <https://www.w3.org/2017/vc/WG/>

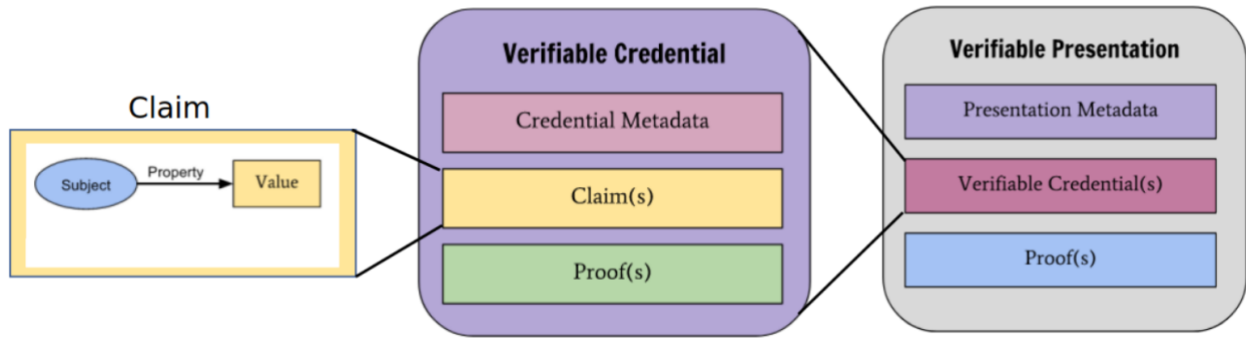


Figure 6: Construction of a Virtual Presentation

Revocation of credentials

VCs can be revoked in anonymously. For this to work, the Issuer creates a long list of numbers that are used to calculate a big number. Each non revoked VC is assigned an index number in that long list and the Issuer publishes the big number. By removing the indexed number from the calculation, a specific VC can be revoked: the Holder can no longer prove they know a number that is used in the calculation of the big number. As the index number is signed by the Issuer in the VC, the Holder cannot falsify the index number and present a different one: the proof requires the knowledge of an Issuer-signed index number in the list of numbers used for calculating the final big number.

As the proof only discloses that the Holder knows a number, but not exactly which number, it prevents linking the Holder, the VC and the presentation of proof, maintaining anonymity.

2.6 Trust framework

According to the Self Sovereign Identity approach, users should be able to create and control their own identity without relying on any centralised authority. The way this is handled has already been explained and consists of an exchange of claims between Issuers and Holders, and between Verifiers and Holders.

Since there is no centralised authority, trust is organised by other means: within SSI, technical trust is built on a decentralised ledger that can prove the existence of (validity of) DIDs and the binding with its corresponding public keys.

What is crucial is that the ledger is well-protected from attempted compromises. Depending on the type of ledger (public or private), there are known risks for ledgers ('Sybil attacks', '51% majority attack', a DDoS attack and 'routing attacks'). Since the ledger is the only source of truth, any compromise is devastating for the SSI. The Sovrin and BCGov ledgers are based on Hyperledger Indy, which is a ledger type using consensus for accepting transactions in a Byzantine Fault Tolerance (BFT) setup. In such a setup, a single node on the ledger is 'in the lead' and suggests new transactions which the other nodes can check and accept. If enough nodes accept the new transaction, it is added to the ledger definitively. There is no incentive to accept faulty transactions like there is for Blockchain or Ethereum, as no active mining takes place to generate income.

As discussed before, DIDs are identifiers and used in combination with Verifiable Claims. In order for a Verifier to obtain information about a user (Subject), the Holder must prove they have ownership of a DID and certain attestations or attributes related to this DID. This proof is based on the cryptographic link between the VC, the DID of the Holder and the Issuer of the VC. The cryptography is set up in such a way that the verifier cannot link the presented proof to the Holder directly: each proof is different (unlinkability). Also, if the Verifier colludes

with the Issuer, the Issuer cannot determine which Holder presented the proof based on the proof itself: the main Holder DID is itself never disclosed to the Issuer or the Verifier.

Trust of the Issuer is established by trusting the Issuer’s DID (e.g., out-of-band, bilateral relationship, trusted lists). The use of a Ledger offers a common truth that includes a record of the DID and its corresponding public key. This way, an Issuer can prove ownership of the DID.

The Verifier can then use the cryptographically protected proof presented to verify ownership and trustworthiness of the claims about the Subject. This proof concerns both Issuer and Holder.

To further understand the roles NRENs and institutions might take in an SSI ecosystem, we suggest further reading of the report “Self-Sovereign Identities”²² published in November 2020 by SWITCH.

The Issuer and the Verifier do not need to trust the Wallet
The Issuer does not need to know or trust the Verifier.

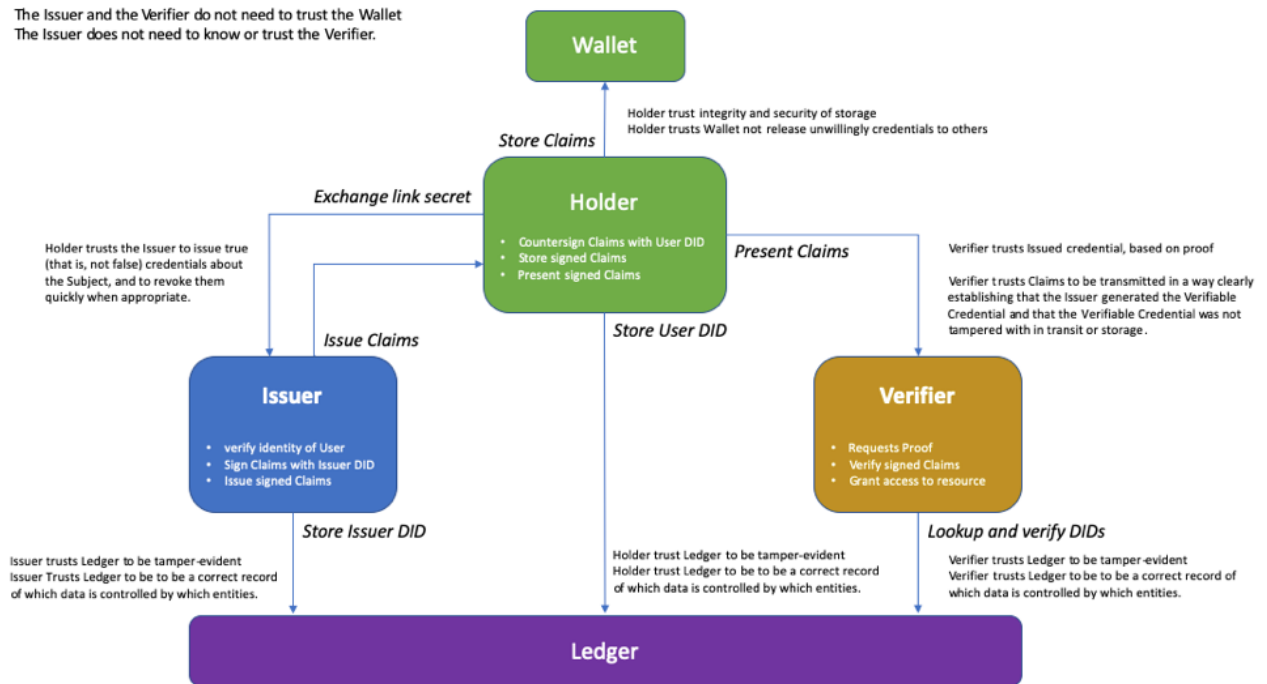


Figure 7: the relationships between the actors in an SSI and the required trust. Note that the link from Holder to Ledger is not used in practice: Holder DIDs are rarely stored centrally.

²² https://www.switch.ch/export/sites/default/about/innovation/.galleries/files/SWITCHInnovationLab_IDAS.pdf

3 Project activities

The SSI project was carried out in the period from September to mid-December 2020. During this period, we carried out a number of activities that are discussed point by point in this chapter.

3.1 Research proposal

The research proposal for the project was prepared prior to the start of the project. This proposal clarifies the research goals and the questions that should be answered. It proposes to undertake functional and technical studies and engage the SSI Community.

The functional studies cover:

- a use case study,
- the current state of Blockchain-SSI
- the relationship between SSI and the GDPR.

The technical studies look into:

- deploying a BC-SSI ecosystem
- conducting field testing of BC-SSI
- combining trust frameworks
- assessing SSI solutions beyond the Blockchain.

In the process of this study, SURF will engage with the SSI community by following relevant developments in W3C and other standards bodies around SSI. It will also work with GÉANT to organise and host an EU NREN workshop on SSI. The complete research proposal can be found in the SURF Wiki²³.

In the four months that we spent on this activity, it quickly became clear that the above list of activities was a rather ambitious.

One of the main challenges the team faced was the fact that much of the SSI-related work is in a very high state of flux. We found out that a lot of the information found on the internet is incomplete or outdated, or no longer aligned with the current technical state of the implementations. This was the case for several of the specifications described in chapter 2.5, as well as for many of the implementations, even for well-known projects like Sovrin.

Given this challenge, we started to review the standards and specifications that underpin SSI. This is described in chapter 3.2. We then decided to focus on finding a product or platform that had sufficient maturity and stability to be deployable by the team. This turned out to be a challenge, as described in chapters 3.3 to 3.6

3.2 Architecture and technology deep dive

The project started with a deep dive into the concept of SSI: the architecture and the technical standards that play a role in SSI. The technical standards 'Verifiable Credentials', 'DID', 'DIDauth' and 'DKMS' were initially studied using desk research. Each team member immersed themselves in one topic and, after a week,

²³ <https://wiki.surfnet.nl/x/tl-aAQ>

explained the technical standard to the other team members. This approach meant that the team was able to master the SSI subject quickly. The pitches also led to discussions about, for example, the maturity and applicability of the standard or the scalability of the solution. These issues had to be sorted out at a later time. The desk research was based on information that can be found publicly on the internet.

The contents of chapter 2.5 are based on this initial research, and additional background information can be found in the SSI basics page on the wiki²⁴.

3.3 Search for solutions

After the team had developed a global picture of the SSI concept and the corresponding technical standards, time was spent on research into existing SSI solutions, implementations of SSI and technical frameworks. This resulted in the following overview of solutions:

Name	URL	Name	URL
Alastria	https://alastria.io/en/home/	LifeID	https://lifeid.io/
BC-gov	https://github.com/bcgov/von-network	P2-ABC	https://abc4trust.eu/
Bitnation	https://tse.bitnation.co/	reclaimID	https://reclaim.gnunet.org/
Blockcerts	https://www.blockcerts.org/	Self-ssi	https://self-ssi.com/en/
Civic	https://www.civic.com/	SelfKey	https://selfkey.org/
Cordentitiy	https://www.corda.net/blog/meet-cordentiy/	ShoCard	https://www.pingidentity.com/en/lp/shocard.html
ESSIF	https://essif-lab.eu/	SORA	https://sora.org
EverID/Everest	https://everest.org/	Sovrin	https://soverin.net/
Evernym	https://www.evernym.com/products/	Trusnet	http://trustnet.fi/
Helixid	https://helixid.io/	TrustChain	https://www.tudelft.nl/technology-transfer/development/
Hyperledger	https://www.hyperledger.org/	TrustID Network	https://www.thalesgroup.com/en/markets/digital-ident/
IDChainZ	https://www.chainzy.com/products/idchainz/	uPort	https://www.uport.me/
Identity.com	https://www.identity.com/	uProve	https://www.microsoft.com/en-us/research/project/u-
IRMA	https://privacybydesign.foundation/irma/	VON	https://vonx.io/
Jolocom	https://jolocom.io/		

3.4 Assessing suitability

It is clear that there are many SSI solutions available. The next step therefore was to create a short list of promising solutions that could fit the objective of the project. In our investigations, we found that:

- a number of SSI solutions actually do not use blockchain technology.
- several solutions offered only a partial implementation. It was important for our purposes that the solution offered a component for the 'Issuer', the 'Verifier' and the 'Wallet'. It was felt that by having all of the components from one implementation, there was a higher chance the components would actually work together in some degree of unison.
- Several implementations were abandoned as they showed no signs of current activity. That would severely hinder our ability to ask questions in case we became stuck or needed support.
- Several implementations used closed source components that were not available to us. As we wanted to fully understand the SSI technology stack both on a functional but also on a technical level, this was deemed unfavourable.
- Most projects lacked complete or up-to-date documentation. While this was an issue for most projects, it was deemed the least problematic if the source code of the implementation was available.

²⁴ <https://wiki.surfnet.nl/x/Ci8iAg>

The overview of the findings on the solutions²⁵ was discussed during several team sessions, resulting in a shortlist of the four most suitable solutions. The shortlist included:

- Alastria,
- BCGov,
- Cordentiy & Identity.com
- the Sovrin Stack.

All of these at first glance provided ready-to-use components and a fair amount of documentation. In addition, the projects seemed active and open for engagement.

These four solutions were then studied in more detail, with the aim to obtain a more detailed description of the solution, the organisation(s) behind the initiative, the degree of activity and liveliness of the initiative, an indication of the maturity of the solution and software, the available SSI functionality (Issuer, Verifier, wallet, ledger, tooling, etc.), the technical standards used and whether the source code was available (possibly Docker with a ready-to-use environment). We also tried a very brief initial technical test of the software provided by these solutions.

As it turned out, the BCGov solution provided rather good documentation and had already implemented a number of (governmental) use cases, both functionally and technically, which we could deploy fairly easily. As this turned out to be based on a Sovrin implementation, we decided not to further investigate the Sovrin Stack as a whole at that point.

During the further investigation into the solution from Cordentiy + Identity.com, it soon became apparent that this solution was not suitable for a technical experiment, as their solutions mostly focussed on supply chain management.

The Alastria project is a Spanish project active in both the government sector and the R&E sector, with fairly good (though outdated) documentation and tooling made available.

It was therefore decided to deploy a working SSI ecosystem with both Alastria and BCGov.

3.5 Platform selection

During an activity spike of two weeks (about eight working days), the Alastria and BCGov implementations were simultaneously reviewed in depth.

3.5.1 Alastria

During the spike, we tried to get the entire Alastria stack working in our own SURF environment. Unfortunately, this did not happen: it took too much time to get the wallet implementation up and running. Unfortunately, the wallet does not work out-of-the box, whereas the BCGov solution offered more confidence that we would be able to get the ecosystem up and running. For Alastria, we made a support call to get us on track. However, it took a relatively long time (in our project span) to get the support question answered, and the response was just a request to file a ticket/issue in the GitHub repository.

To try to get the wallet working, a private development environment was set up. From looking into the source code, it appeared that at least a hard-coded server URL was included. This server was no longer operational

²⁵ <https://wiki.surfnet.nl/x/My8iAg>



and the configuration would have to be adapted to the SURF environment anyway. Unfortunately, even with a number of adjustments of the software, the wallet could not be made to work.

Additionally, the Alastria setup uses a construct of Smart Contracts on the Ethereum blockchain that seem to result in a setup in which the anonymity and unlinkability aspects are less important. The application of Alastria appears to be in the logistics area, where all parties are known upfront and actual anonymity is less of a concern. With respect to the alternative implementation of BCGov/Sovrin/Hyperledger, this would lead to an inferior solution for our use cases.

It was therefore decided not to invest extra time in trying to get the Alastria wallet up and running, but instead to instead invest this time in the BCGov implementation.

The following has been achieved for the Alastria stack: the ledger has been installed locally, the Alastria libraries that interact with the ledger have been tested, the development environment for the wallet has been set up (documentation on this has been donated to the Alastria project).

3.5.2 BCGov

The BCGov's project is driven by Canadian government agencies (British Columbia, Ontario and Public Services Procurement Canada). It is an open-source solution based on Hyperledger / Indy ledger and an implementation of the Sovrin framework.

The source code of BCGov can be found in the various GitHub repositories, and there are many adjacent repositories for various components and example implementations (using Docker). A description is available for the architecture of BCGov and the technical standards DID, DIDcomm and Verifiable Credentials and can be used with wallets from other parties, such as the Trinsic Wallet, esatus Wallet or the Lissi Wallet. See chapter 4 (*'Testing the BCGov stack'*) for more background information on this stack.

The Hyperledger Indy ledger solution works in concert with the Aries Agent-to-Agent communication protocol. Both implementations go to great lengths to ensure that all activities are cryptographically signed and verifiable, and that the anonymity of Holders is guaranteed. The latter is done using cryptographic techniques such as link secrets, blind signing and revocation using tail files. An in-depth description of these techniques goes beyond the scope of this document, but information about them is readily available on Wikipedia. An important figure in this area of cryptography is Jan Camenisch, who wrote several leading papers on anonymous cryptography. Hyperledger makes extensive use of Camenisch-Lysyanskaya (CL) signatures to allow blind signatures and prevent the leak of information.

BCGov has a public (beta) service available based on this implementation called VON²⁶ (Verifiable Organizations Network), which underpins "OrgBook BC"²⁷, the digital equivalent of a register of companies.

Based on our initial experiences and the significantly better quality and availability of documentation, we decided to continue our experimentation with the BCGov solution.

3.6 Setting up the infrastructure

A development environment and a test environment were set up. These both consist of a number of virtual hosts that host Docker containers. The BCGov solution is highly compartmentalised: every component is run in a separate Docker container. It took some time to figure out all relations and the required ports and firewalls to

²⁶ <https://vonx.io/>

²⁷ <https://orgbook.gov.bc.ca/en/home>

be opened between these Dockers. The complete BCGov ecosystem is operational in SURF's technical environment, and the network and the components have the following structure.

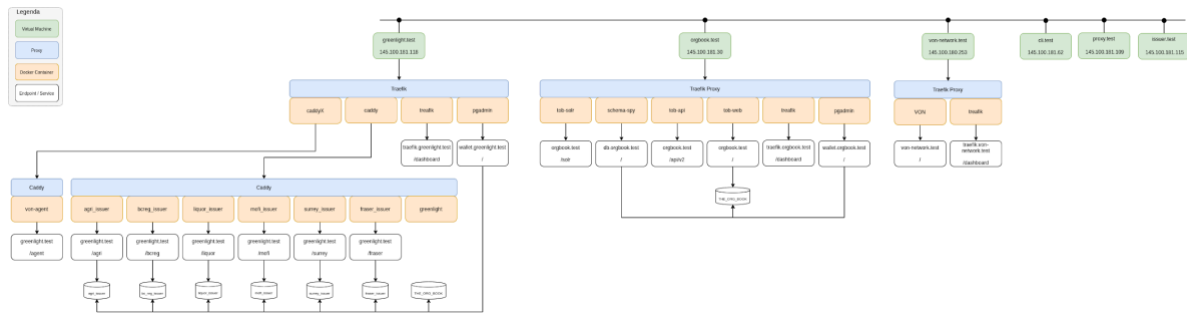


Figure 9: schema of the BCGov ecosystem in SURF's technical environment

Development is carried out on local laptops and the code is committed to the SURF Gitlab environment. In order to manage and deploy the ecosystem, an Ansible playbook has been created. Chapter 5.5 ('Deployment') describes the details of this. The team uses Trello and the SURF Wiki to order the activities and log findings.

3.7 Building out the experiment

In order to gain a better understanding of the standards and technology used with BCGov, it was decided to create an application similar to the BCGov Greenlight demos. For details, see chapter 5 ('Implementation of SURF SSI services'). This application can be used for testing purposes later on. This activity resulted in an application that interacts with the Ledger and can act as both Issuer and Verifier: it supports creating new credentials based on defined schemas and credential definitions ('cred defs') and sending those credentials to a connection. We also made a simple implementation to create a proof request, allowing the user to request a proof consisting of one or more attributes of an existing schema or cred def from the wallet connected to the current session. Details on that experiment are documented in the chapter 'Implementation of SURF SSI services'.

3.8 Define use cases

Part of the research proposal is to carry out functional studies, aimed at collecting (national) use cases for SSI-based technology in R&E. During the project, an initial set of use cases was described. For each use case, we have described the global goal, stories, actors and pre and post conditions. The use cases will be helpful in testing the 'SURF SSI Services' and may inspire business owners considering using SSI in their own business processes.

The use cases are:

- **Obtaining claims**
Users want to obtain claims from their university acting as an Issuer. Users therefore log in at the university using a traditional login and are issued a VC to be stored in the user's wallet.
- **Interaction with Virtual Organisation (VO) Membership Management Service**
A user wants to access resources of an application from the VO. The user therefore needs to access the Membership Management Service of the VO in order to obtain a 'membership' claim. That claim grants access to VO resources.
- **Create a Curriculum Vitae**
A user wants to create a (formal) curriculum vitae online. In order to create this CV, accreditations

from various sources (Issuers) must be combined and presented as a whole in a CV that a third party creates online.

- Interaction with a service (Verifier)
Users want to interact with a service to provide proofs in the form of credentials from a wallet.

3.9 Documentation of findings

As a final activity, the project team documented the findings on SSI, as can be found in this document and the SSI wiki²⁸.

²⁸ <https://wiki.surfnet.nl/display/SSI/Self+Sovereign+Identity+Home>

4 Introducing the BCGov stack

In this chapter, the BCGov components are explained in more detail. The applications provided by the government of British Columbia fall into three categories:

- the Ledger,
- The OrgBook and
- the Greenlight Demos.

While the government of British Columbia is running a beta instance of OrgBook and their ledger publicly, we are under the impression the setup is meant to be an experiment to learn how SSI technology may be leveraged in the context of governmental use cases.

4.1 Ledger

BCGov implements a basic Hyperledger Indy blockchain solution²⁹, using a collection of nodes based on Hyperledger Indy and Hyperledger Plenum, with a blockchain explorer web interface. Indy is the basic ledger node³⁰, while Plenum³¹ is the node-to-node communication layer wrapping Indy specific ledger messages in a common Byzantine Fault Tolerant messaging protocol.

The BCGov version is based on a management repository that is maintained on GitHub³². This repository serves scripts and documentation to create a small local network of ledger nodes. It also has a very basic, Vue-based web interface with a simple Python back-end to browse through the ledger transaction and perform simple searches.

Within the SURF project, work was done to implement these scripts and arrange the deployment through an Ansible playbook, allowing a more flexible configuration of a variety of deployment models³³.

The Hyperledger Indy implementation has several trust roles that govern the way the Ledger is managed. Each identity on the Ledger can have a role attached to its DID. There are four basic roles, in descending order of operational power:

- Trustee
- Steward
- Endorser or Trust Anchor
- Anonymous

The *Trustee* role can add new nodes, new stewards and sign any kind of transaction on the Ledger. The *Trustee* keys are supposedly in the hands of a board of trusted high-level administrators that govern the ecosystem. Their most important task is to onboard *Stewards* for the day-to-day operations.

The *Steward* role is attached to the maintenance of a specific node or nodes in the network. The *Steward* has the right to sell transaction credits to users as a means to finance the network. Users spend these transaction

²⁹ <https://www.hyperledger.org/use/hyperledger-indy>

³⁰ <https://github.com/hyperledger/indy-node/>

³¹ <https://github.com/hyperledger/indy-plenum>

³² <https://github.com/BCGov/von-network.git>

³³ <https://gitlab.surf.nl/ssi/deploy-playbook>

credits for specific ledger transactions following a ledger-wide pricing table. Stewards can also add new users and new *Endorsers* to the ledger by creating an appropriate transaction.

The *Endorser* or *Trust Anchor* role is an intermediate role. It is supposed to be a role given out to trusted institutions that can then create non-privileged user DIDs on the ledger, issuer Schemas and Credential Definitions, etc. This role would fill the domain part of the ledger with meaningful information. For each transaction they perform, they need to pay from their previously allocated transaction credit balance. Transactions are priced differently, with a new schema costing much more than a simple credential revocation transaction, even though the transaction itself is not larger in any meaningful way.

The last role is that of a simple user with a DID registration. This user can be anyone or anything, including a non-trusted organisation or institution. All transactions that this user wants to perform need to be endorsed by an *Endorser* or higher-level role. As the *Endorser* needs to pay for the transactions, they are motivated to check the user-requested transactions for applicability, conformity and validity.

4.2 The OrgBook

The OrgBook (TOB) is an application offering company registry-like features. The application consists of a separate Angular-based front end interacting with a specific API back end for large search operations on the OrgBook database. This is supported by an Apache Solr database. The application is essentially a yellow pages-like search form. Adding new entries to the TOB database is done by back-end calls to the API layer. There is no administrative user interface.

The implementation consists of several separate Docker containers (web interface, API layer, Solr, RabbitMQ, database, cloud agent) and supports additional scaling. The original code for the first iteration of TOB can be found at GitHub³⁴.

Since then, the TOB code has been pushed to the Hyperledger group as 'Aries-VCR' (Verifiable Credential Registry)³⁵.

Both applications deliver roughly the same user experience. Both stacks were deployed on separate servers, linking to the same ledger implementation. Work was done to convert the managing scripts into an Ansible playbook for easy deployment.

4.3 Greenlight demos

BCGov created a set of demo applications under the name 'Greenlight'³⁶. These applications each run in a separate Docker container (with a separate wallet database container) and are based on a common framework implemented by the Python packages VON-X³⁷ and VON-anchor³⁸. These packages are basic implementations of the various calls into the libindy Python wrappers, along with additional functionality where the libindy Python wrappers fall short.

³⁴ <https://github.com/BCGov/TheOrgBook>

³⁵ <https://github.com/BCGov/aries-vcr>

³⁶ <https://github.com/BCGov/greenlight.git>

³⁷ <https://github.com/PSPC-SPAC-buyandsell/von-x.git>

³⁸ https://github.com/PSPC-SPAC-buyandsell/von_anchor.git



The Greenlight demos are effectively a set of Yaml specifications that instruct the common web server framework to create a specific interface with schemas, credential definitions and revocation interfaces.

They also provide services for testing (proving) your credentials, so that a whole chain of use cases can be demonstrated.

These demos run without an intermediate cloud agent and implement all necessary calls and message handling through an abstracted asynchronous service layer in VON-Anchor and VON-X. Because of this, the actual implementation can be considered 'mildly outdated' and does not support newer features from the rapidly changing SSI landscape.

4.4 Messaging ecosystem

Orthogonal to the basic ledger ecosystem, the messaging ecosystem encompasses all communication between Issuer, Holder and Verifier. In theory, they can communicate with any ledger implementation and currently the Hyperledger Aries protocol³⁹ actually uses the public keys (*verkey*) instead of the DID and DID methods defined in various specifications⁴⁰ for resolving public keys (which are ledger-constrained). The messaging ecosystem distributes all actual (trusted) content: credentials, revocation data, requests for credentials and proofs. Only the basic framework (schema, credential definition, DIDs) are publicly and permanently made available on the ledger.

Whenever an Issuer or Verifier wants to interact with a Holder, it offers the Holder an invitation to communicate using a messaging system called a *connection*. This connection is established by the Holder by visiting an Issuer-specific URL that contains encryption information for further communication. This is generally done using regular HTTP(S) API interfaces using ledger-specific mobile device applications called wallets. These wallets are linked to cloud agents that can communicate with Issuer and Verifiers using a back-channel connection cloud agent and Issuer/Verifier servers directly, in parallel to the browser interface the user is using.

³⁹ <https://github.com/hyperledger/aries-rfcs>

⁴⁰ <https://www.w3.org/TR/did-spec-registries/#did-methods>

5 Implementation of a SURF SSI demo

In order to become better acquainted with the underlying technology, it was decided to create a SURF Ledger based on the ledger deployment available as part of the BCGov implementation. Next, an application similar to the Greenlight demos was developed. The goal was to implement an Issuer and Verifier that interact with the SURF Ledger, thereby creating rudimentary SURF SSI services. While the Greenlight demos were functional after deployment, most of these were targeted at governmental use cases, such as driving licence or drinking permit issuance. By writing our own deployment, this would force us to become intimately familiar with the ledger and the interaction between the various components and protocols, while at the same time working towards implementing our own SURF use cases. This section documents the lessons learned from that experiment.

5.1 Ledger setup

Deploying the ledger proved to be very straightforward. By default, the BCGov deployment starts four docker containers, each containing one ledger implementation. These four distributed ledgers keep each other in sync. Later on (see chapter 5.7), we optimised the deployment by separating out various components with the use of Ansible, and also tested dynamically adding and removing nodes to the ledger.

5.2 PHP-based Issuer/Verifier

To build on the tradition of introducing new tools and new systematics within the context of SSI, Hyperledger and BCGov, an initial implementation of SURF SSI services was made using PHP (Laravel). Because of the absence of a PHP module or PHP wrappers in the libindy SDK, the ledger interface was written using a pipe construction into the command line interface (libindy-cli). As all basic ledger commands can be constructed using the CLI interface, the blockchain part of the SSI solution could be implemented using available documentation.

Although successful in the beginning, the limitations of libindy-cli quickly became apparent and this solution was abandoned. The CLI interface has very limited support for the cryptographic tools required for creating credentials and proofs and for verifying signatures and keys. As the underlying keys are not readily available, but kept in a secure wallet location, the lack of cryptography inside the CLI itself limits the usability of the CLI as more than a ledger interface tool.

5.3 Setup using ACA-Py

A second attempt was made by leveraging the Aries-Cloud Agent-Python (ACA-Py⁴¹) framework. This tool is also used in some BCGov products and serves as a cloud agent interface to both the ledger and external holders. The Python-based framework serves as a test bed for most Agent-to-Agent development in the current Hyperledger ecosystem.

The initial idea was to create a front-end tool that uses an API to interact with ACA-Py without the need for its own database. ACA-Py has support for maintaining and querying lists of schemas, credentials, connections, etc. Although this looked promising, the limitations of ACA-Py in releasing internals and keeping some sort of client-side state were indicative of the function of ACA-Py as a cloud agent and not as an edge-agent back end.

⁴¹ <https://github.com/hyperledger/aries-cloudagent-python>

5.4 Setup using Django/React

Based on the findings with the ACA-Py solution and the initial setup, a new platform was developed based on a Django REST framework back end and a REACT SPA front end. The Python back end could then easily integrate with the Python wrappers of the LibIndy-SDK. Although those Python wrappers do not implement 100% of the LibIndy-SDK functionality, they open enough of it for the purposes of this tooling.

After some initial reimplementing of the original PHP code from the initial setup, concerning the creating of DIDs, schemas and cred defs on the ledger, the correct implementation of an Agent-to-Agent messaging interface caused problems with the available Trinsic and Lissi wallets: connections could not be properly established, and the reasons why were not made apparent by either wallet implementation. Instead of spending a large amount of time getting this right, a decision was made to reintegrate ACA-Py.

5.5 Setup using ACA-Py with Django/React

The final setup uses the Django/React framework of the previous implementation, but replaces part of the code by relevant calls to an intermediate ACA-Py instance. This quickly showed that ACA-Py was indeed able to properly connect to Lissi and Trinsic wallets.

The downside of this implementation is that ACA-Py maintains its own keys and requires the controller application to create schemas, cred defs and revocations registries through ACA-Py. It is not possible to keep the ledger-specific elements outside of ACA-Py, but retain the possibility of sending Agent-to-Agent messages.

Within this setup, functionality was created to create a schema and associated cred defs (with theoretical support for revocation). A system was developed to create new connections using a QR code, or to import an old, existing connection into the current web server session by sending a connection code. Efforts at implementing DIDAuth as envisioned at the beginning failed. It seems that there is currently no easy way of indicating to an Issuer or Verifier that you wish to connect using an existing DID and connection. The wallets do not offer that feature and the DIDcomm messages do not support such a generic request.

The implementation also supports creating new credentials based on defined schemas and cred defs and sending those credentials to a connection. Furthermore, a simple implementation to create a proof-request was made, allowing the user to request a proof consisting of one or more attributes of an existing schema or cred def from the wallet connected to the current session.

An overview of the application at the time of writing and the various parties, platforms and applications involved in the overall functionality is shown in Figure 10.

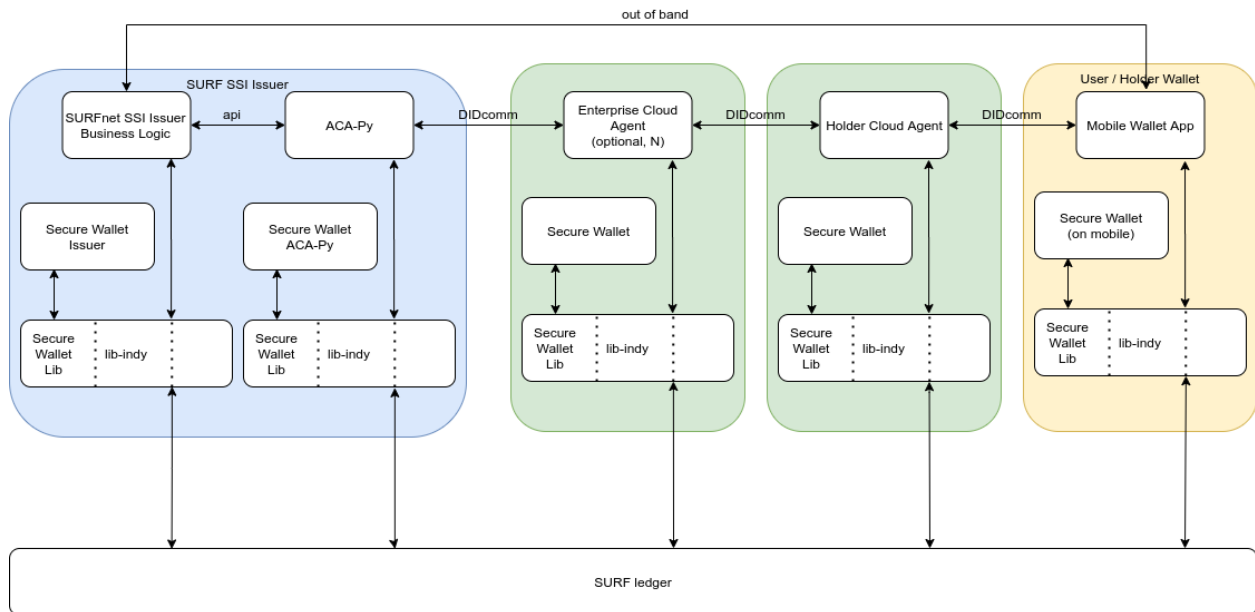


Figure 10: overview of the SURF application

The Issuer application in the blue square consists of two separate parts: the business logic and the ACA-Py agent. These communicate using an ACA-Py specific API. The ACA-Py agent can then communicate with other entities using DIDcomm protocol. The green parts are cloud-based entities, while the yellow part is the installation on a mobile or desktop platform (edge agent).

In a standard situation, the end user has a wallet application on his/her phone. This wallet is connected at installation time to a cloud agent of the wallet supplier. Between the latter cloud agent and the ACA-Py cloud agent, one or more enterprise cloud agents might be present to route traffic. These intermediate cloud agents are mentioned in the DIDDoc of the DIDs presented by the Holder and document the routing chain for messages intended for the Holder.

The Holder and the SURF SSI Issuer app initially communicate using out-of-band communication, which takes the form of browser traffic (the front-end website) and QR codes. The QR codes can be scanned by the Holder to insert a new activity into the wallet. In most cases, the QR codes are shortened URLs that link to a redirection (HTTP 302) that resolves to a much longer URL containing a base64 encoded DIDcomm message.

5.6 Information flows

UML diagrams have been drawn up for the main processes of the SSI implementation. These are the processes of creating a connection, credential issuing and credential revocation.

The following image depicts the process of creating a new connection between the Issuer application and a user (Alice). This process works in the same way when creating a new connection between a verifier or another agent (e.g.: Bob). One agent initiates the connection request, and this initial request is transported using an out-of-band mechanism (QR code, clicking a link, etc.). Note the absence of the use of the common ledger in this protocol. All key information is transferred within the protocol and no trust is created based on the transmitted information alone. This protocol merely creates a secure communications channel over the agent-

to-agent messaging backbone (DIDcomm). The Issuer CA in this picture is the ACA-Py framework. As can be seen, it also performs actions that may be considered out of scope for a mere cloud agent relay.

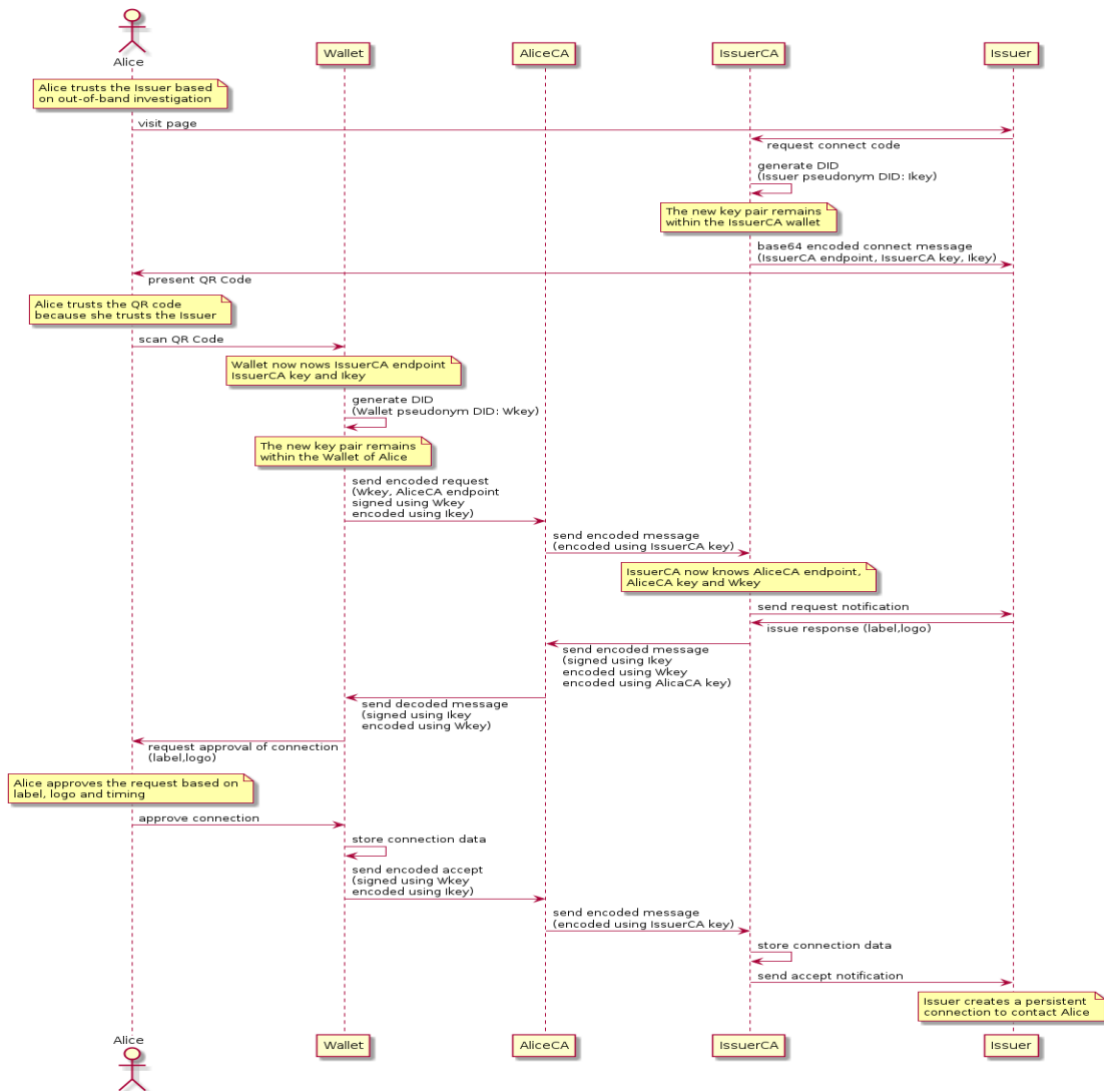


Figure 11: Process of creating a new connection between the Issuer application and a user

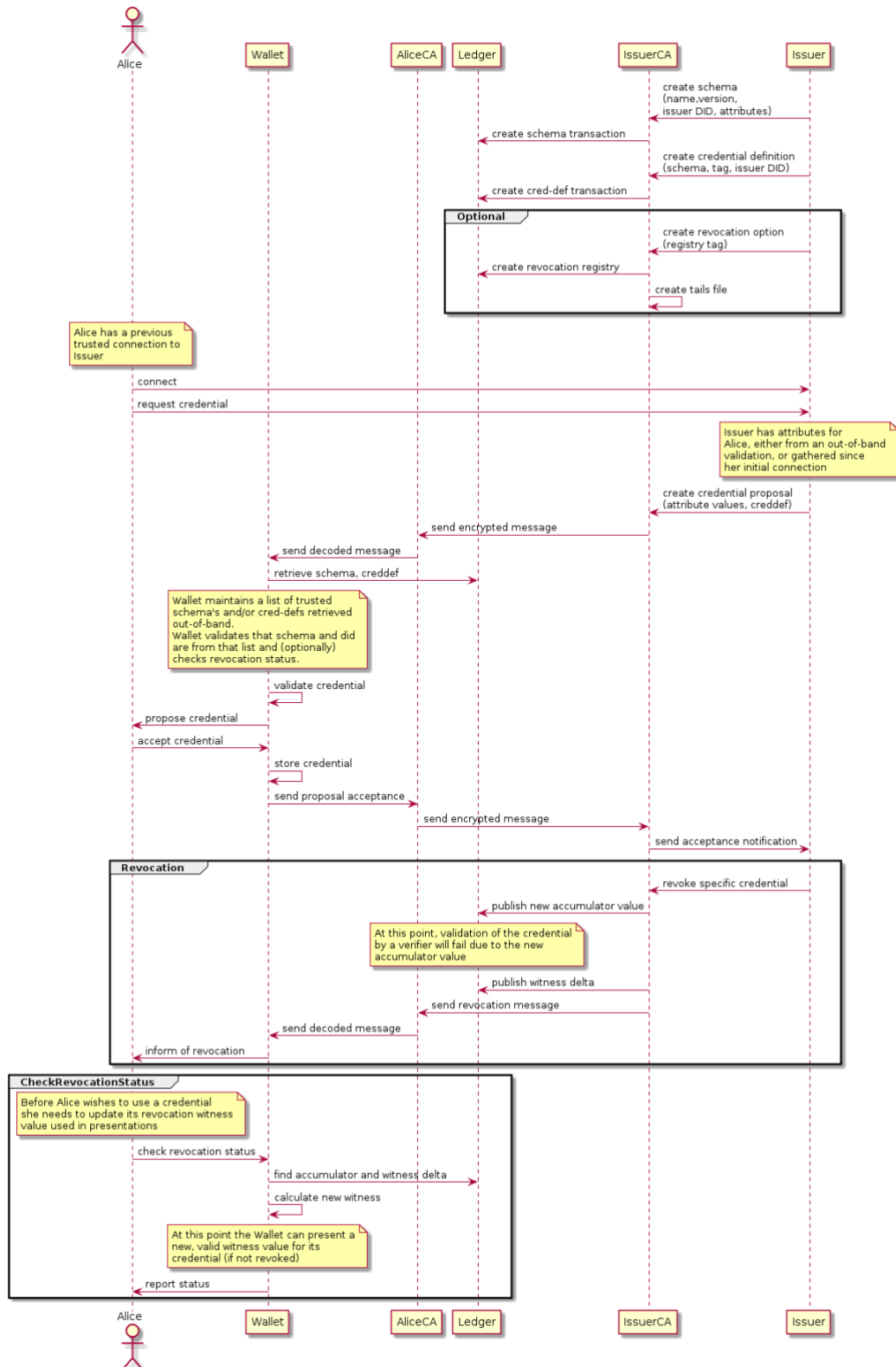


Figure 12: process of credential issuing and revocation

Credential issuing, as depicted in the previous image, is initiated by the Issuer. However, a preceding activity by Alice is usually required: authenticating at the Issuer, creating a new connection or reusing an existing one and,

finally, initiating the process for delivering a credential. This is not absolutely required as long as the Issuer has a valid connection to Alice. It can push a new credential over this connection at any time.

Credentials come in two flavours: without revocation and with revocation. Revocable credentials require a few additional transactions on the ledger to manage a revocation registry. The steps required for revocation by the Issuer and for checking the revocation status by the Holder are also specified in the image. Checking the revocation status should be done immediately before a credential is presented for proof.

The image below shows the latter sequence in a more abstract way:

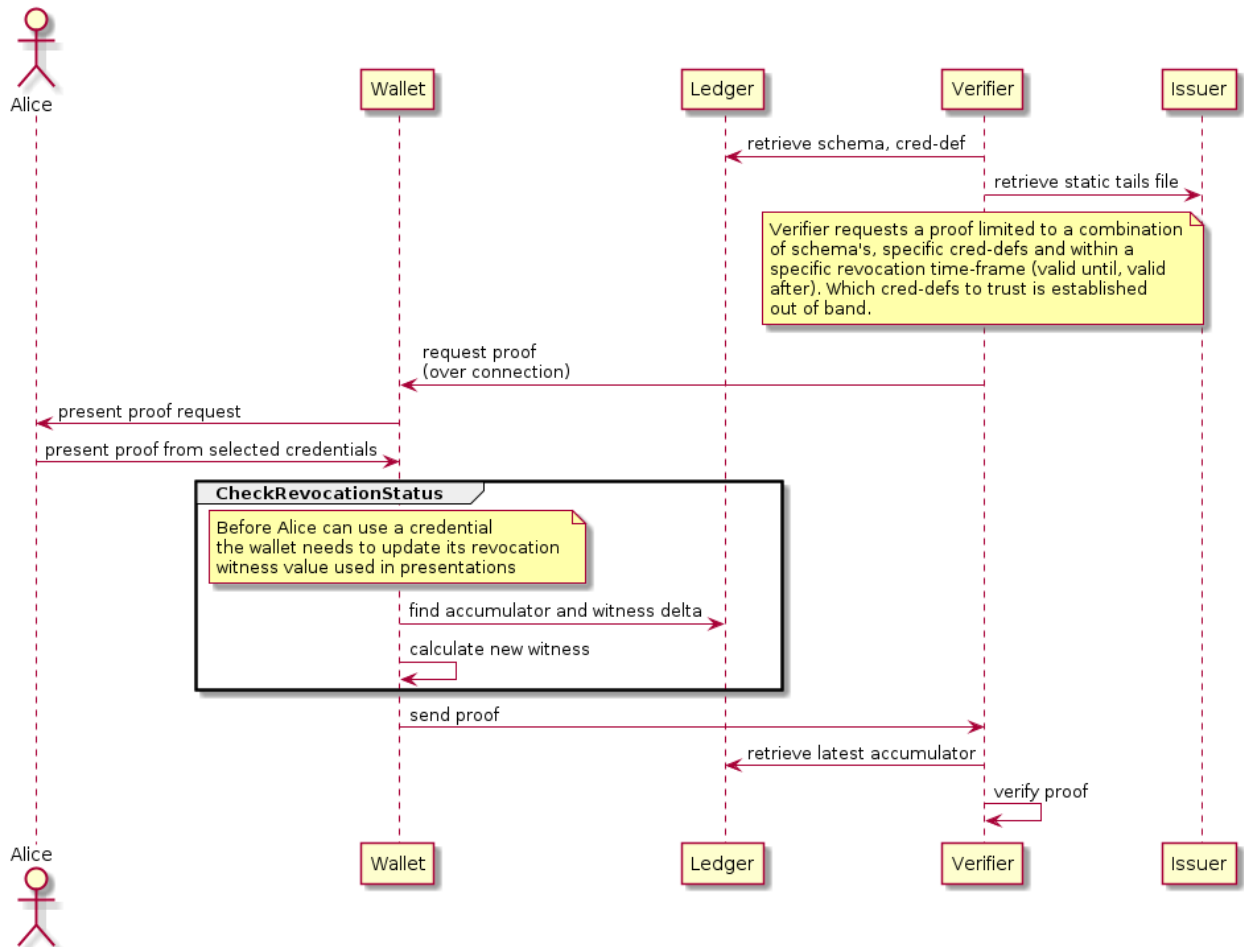


Figure 13: process of credential revocation

In this situation, the Issuer initiates the proof capabilities by first retrieving the information about the credentials and schemas for which it wants to receive attributes or predicates (proof over values of an attribute). For revocable credentials, the Issuer can retrieve the static *tails file* from the URL published in the Ledger, but hosted by (or under supervision of) the Issuer. The Verifier then uses an existing connection to request the proof.

The Holder selects credentials from its wallet that match the requested proof, based on restrictions on the attribute origins set out in the Issuer's proof request.

If any of the selected credentials can be revoked, the Holder should check the revocation status or risk presenting a revoked credential. That by itself is not an error: a revoked licence may still be a valid credential in

some circumstances. Whether the underlying frameworks still present attributes of revoked credentials or return a generic proof failure error was not tested.

To verify the proof, the Verifier does not need any additional information from the ledger, except in the case of revocable credentials.

5.7 Deployment

The infrastructure we have developed is deployed via Ansible. This way, we have control over the deployment of the components during future follow-up activities. Using Ansible helped us in tweaking the software components from open source, since they have their specific requirements and interdependencies.

We have used several VMs to host the various components. For both the development and the test environment, four VMs were needed. The OrgBook from BCGov uses a lot of machine resources. That is why we have split different components across different machines.

As a standard procedure, our machines are shielded behind eduVPN. However, some secure public access to services is needed because the mobile wallet needs to communicate with our Ledger and issuer agent. We have therefore opened all our services via a HTTPS proxy.

The Ansible playbook for the infrastructure: <https://gitlab.surf.nl/ssi/deploy-playbook>.

The playbook delivers following (main) functionality:

Roles	Delivers
Traefik ⁴²	Let's Encrypt SSL certificates and service proxying
Ledger	This is our central ledger based on the Hyper Indy VON-network ⁴³
OrgBook	BCGov OrgBook ⁴⁴
Agent	A basic template for VON network agent ⁴⁵
IssuerVerifier	Issuer + cloud agent as required for the ACA-Py / Django demo application

⁴² <https://doc.traefik.io/traefik/>

⁴³ <https://github.com/BCGov/von-network>

⁴⁴ <https://github.com/BCGov/TheOrgBook>

⁴⁵ <https://github.com/BCGov/von-agent-template>



IDP	SimpleSamlPhp for SAML interaction ⁴⁶
Greenlight	The set of agents used within the OrgBook demo. ⁴⁷

Figure 14: Deployment Playbook Roles

5.7.1 Genesis file

A genesis file is an important file that reflects the existence of a given Ledger. The file contains a reference to the initial ledge validation nodes with their public keys. If a wallet is to be connected to the ledger, that wallet typically will need to consume the ledger genesis file.

For example, this is the content of the ledger started with three validator nodes. The genesis file for the SURF implementation can be found at: <https://von-network2.test.ssi.lab.surf.nl/genesis>

```

{"reqSignature": {}, "txn": {"data": {"data":
{"alias": "Node1", "blskey": "4N8aUMH5gJ0Vgkpm8nhNEfdFtXHznoYREg9kirmJrkiVg4oSEimFPfnsQ6M41QvH2Z33nves5vfn9nUwNFJBYtWVnHYMATn76vLuL3zU88KyeAYChfsh3He6UHCXDcaecHVz6jhCYz1P2UZn2bDvruL5
wXpgh8fBaLkm3Ba", "blskey_pop": "RahHYiCvoNCTPTrVtP7MCS5eYrsUA8WjXbdhNc9dehlagE9bgiJxWBXYNFbnJXoXhMFMVYqghRog737YQemH5ik9oL7R4NTTCz2LEzhkgLzB3QR0qJy8Yv7acbdhRAT8n09ULDaVL9NBpnKBXW4
LEMePaSHw6RzPndAX1", "client_ip": "145.100.180.160", "client_port": "9702", "node_ip": "145.100.180.160", "node_port": "9701", "services":
{"VALIDATOR"}, "dest": "0w6pDLhcBcoQesN7zqfot7gPa7cbuq2pkX3Xo6pLPhv", "metadata": {"from": "Th7MgTaRzVRYnPiabds81Y", "type": "0", "txnMetadata":
{"seqNo": "1", "txnId": "fea82e10e894419fe2bea7d96296a646f50f93f9eeda954ec461b2ed2950b62", "ver": "1"}
"reqSignature": {}, "txn": {"data": {"data":
{"alias": "Node2", "blskey": "37rAppXVoxzKhz7d9gkUe52XuXryuLXoM6P6LlWDB7LSbG62Lsb33sfG7zq8TK1MKwuchj1FKNzVpsnafmGLvXN88rt38mNfs9TENzm4QBd8zsvCuoBnPH7rpYYDo92DNJePaDvRvqJKByCabubJz3XXKbEe
shzpz4Ma5QYpJqjk", "blskey_pop": "Qr658mW22YC8JGXWMDQTzu2CWF7NK9EwXphGmcByChfybUuLxbG65nsX4JvD4SPNtkJ2w9uglyLTfj6fmuDg41TgECCjLCij3RMsv8CwevBvGVN67wsA45DFWvqvLtu4zjNnE9JbdPtc124WCPA3Xan44
KlHOHAg9EVeaRys8zoF5", "client_ip": "145.100.180.160", "client_port": "9704", "node_ip": "145.100.180.160", "node_port": "9703", "services":
{"VALIDATOR"}, "dest": "8ECVsk17mjsjKRLiQtssMLgp6EPhWxtaYStWPSGAb", "metadata": {"from": "EbP4yNeThL6q385GuVpRV", "type": "0", "txnMetadata":
{"seqNo": "2", "txnId": "1ac8aee2a18ced660fef8694b61aac3af0ba875ce3026a160acbc3a3af35fc", "ver": "1"}
"reqSignature": {}, "txn": {"data": {"data":
{"alias": "Node3", "blskey": "3WfPdbg7C5cnLYzWfZevJghubkFALBfCBok15GdrKMUhUjGsk3jV6QKj6MZgEubF7oqCafXndkm7eswgA4sdKTrc82LLGz2Bd6vNqU8dZup6uYuf32KTHPQbuUM8Yk4QFXjEf2Uu2JcNkdgppeUSX42u5
LqddDpSNWUK5deC5", "blskey_pop": "Qr658mW22YC8JGXWMDQTzu2CWF7NK9EwXphGmcByChfybUuLxbG65nsX4JvD4SPNtkJ2w9uglyLTfj6fmuDg41TgECCjLCij3RMsv8CwevBvGVN67wsA45DFWvqvLtu4zjNnE9JbdPtc124WCPA3Xan44
HvDtk6PFKfVcVxYrNYjh", "client_ip": "145.100.180.160", "client_port": "9706", "node_ip": "145.100.180.160", "node_port": "9705", "services":
{"VALIDATOR"}, "dest": "DKVxG2FXXTU8yT5N7hGEBXB3dfdAnYv1JczDUBpmDxya", "metadata": {"from": "4cU41vWw82ArfxJXhKzXPG", "type": "0", "txnMetadata":
{"seqNo": "3", "txnId": "7e9f355dffa78ed24668f0e0e369fd8c224076571c51e2ea8be5f26479edebe4", "ver": "1"}
}
    
```

Figure 15: Contents of a Genesis File

The Genesis file contains artifacts of the Validation Nodes, for example:

- Alias, for example “Node1”
- Blskey: public key of the node
- Blskey_pop: proof of possession of key
- Node_ip: the IPv4 address of the node⁴⁸
- Node_port: the port at which this node is listening. Nodes are listening to other nodes and replicate domain data between each other.
- Services: The role of this node, “VALIDATOR”
- From: The DID of the Steward⁴⁹ that initiated this node

5.7.2 5.7.2. Distributing the Ledger

The BC-GOV setup that we have used by default comes with an arrangement in which all ledger components are deployed on a single machine. The Docker containers are started simultaneously and are connected to the

⁴⁶ <https://github.com/OpenConext/OpenConext-DIY>

⁴⁷ <https://github.com/BCGov/greenlight>

⁴⁸ We have looked into the option to use a FQDN instead of an IP4 address, however this is not allowed.

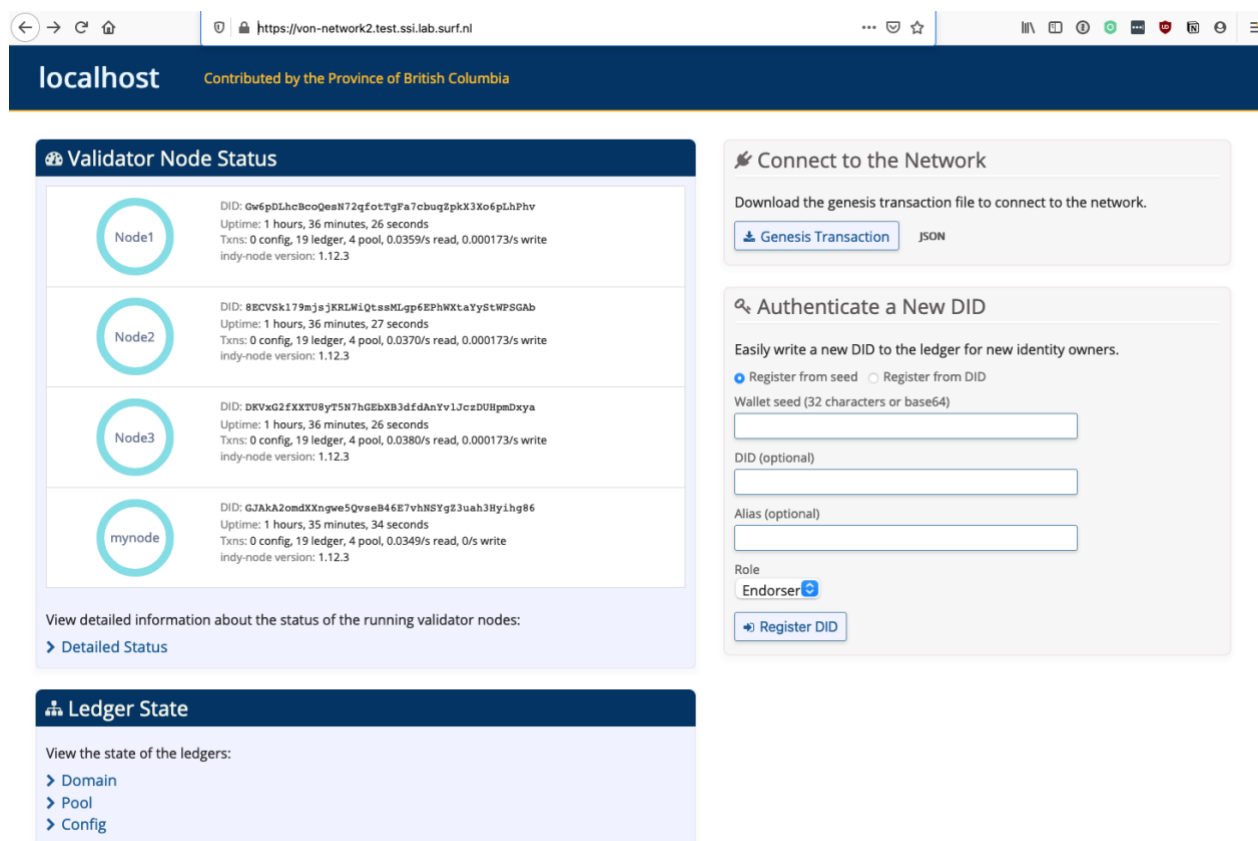
⁴⁹ The hierarchy is that the Trustee (Ledger Origin) adds Stewards. Only Stewards can create a Validator Node.

same Docker network bridge. The standard configuration of BC-GOV comes with a single webserver (a monitoring dashboard for the ledger) and four ledger nodes.

Since the ledger should fundamentally support scalability and distribution, we have also evaluated additional configurations:

- Bringing up the initial ledger with only three nodes
- Starting up another new node and adding this node to the running ledger
- Verifying that all nodes run happily together and synchronise with each other
- Stopping and restarting the ledger
- Stopping and restarting a node

The detailed steps to get these configurations to run without errors and conflicts took quite some substantial exploration. The required recipe is documented in the Ansible playbook⁵⁰.



The screenshot shows a web browser displaying the BCGov Dashboard. The browser address bar shows `https://von-network2.test.ssi.lab.surf.nl`. The dashboard has a dark blue header with 'localhost' and 'Contributed by the Province of British Columbia'. The main content is divided into several sections:

- Validator Node Status:** A table listing four nodes: Node1, Node2, Node3, and mynode. Each node entry includes its DID, uptime, transaction statistics (Txns), and the Indy-node version (1.12.3).
- Connect to the Network:** A section with a 'Download the genesis transaction file to connect to the network.' message and a 'Genesis Transaction' button labeled 'JSON'.
- Authenticate a New DID:** A section with the heading 'Easily write a new DID to the ledger for new identity owners.' It offers two options: 'Register from seed' (selected) and 'Register from DID'. It includes input fields for 'Wallet seed (32 characters or base64)', 'DID (optional)', and 'Alias (optional)'. A 'Role' dropdown is set to 'Endorser', and a 'Register DID' button is visible.
- Ledger State:** A section with the heading 'View the state of the ledgers:' and three expandable links: 'Domain', 'Pool', and 'Config'.

Figure 16: BCGov Dashboard to visualise the ledger, domain, pool, config, genesis file and running nodes

One interesting finding during this experiment is that the genesis file holds the configuration of the ledger during its initial start. Starting with the three initial nodes, it contains the public key details of these three nodes.

⁵⁰ https://gitlab.surf.nl/ssi/deploy-playbook/-/blob/develop/roles/ledger/templates/add_node.sh.j2

After node 4 is added to the configuration (named “mynode” in Figure 16), the genesis file is not changed in any way. If extra nodes are added to the configuration over time and the original nodes are retired from the configuration, the genesis file will keep references to these nodes which would then no longer exist. This raises the question of how a ledger should survive these kinds of configuration adjustments. From our current knowledge and experience with the Trinsic Wallet Application, a remote wallet will connect to the IPv4 and node-port addresses listed in the ledger genesis file so these IPv4/port addresses must be preserved over time.

To operate a ledger, the following internet accessible components need to be in place:

- A webserver, serving dashboard functionality and the genesis file
- For each node, one IPv4 address and two TCP ports

5.7.3 Traefik

As already said, the BCGov deployment initially deploys all components of the same virtual machine and uses many different ports to expose the various applications. In addition, none of the applications support the use of https by default.

For all components that we have experimented with and which are serving HTTP services, we therefore put them behind a reverse proxy. Traefik provided this functionality and added to the Ansible playbook. Once the Traefik role is executed from the runbook, the reverse proxy is operational and additional HTTP services can be hooked up to the reverse proxy server very easily. Traefik will take care of SSL handshakes and automatically requests and renews Let’s Encrypt certificates.

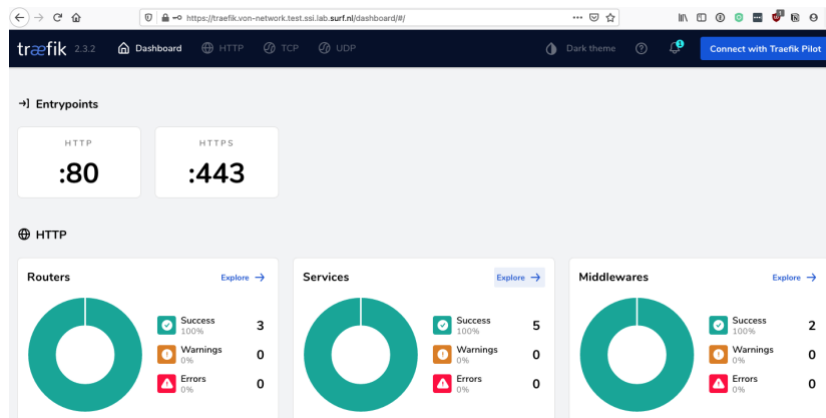


Figure 17: Traefik dashboard

By using Traefik, we were able to easily enable https support for all http-based components, and we found no interoperability issues with the various components of the setup when we enabled https.

6 Findings

This chapter describes the findings of testing with the BCGov SSI implementation. These are findings from both the SSI concept as well as this implementation which is based on the Sovrin platform.

6.1 Use cases

We tested the use cases defined in chapter 3.8 and found we could complete all of them:

- **Obtaining claims**
As shown by using our demo platform we deployed a protocol translation tool to turn SURFconext into an SSI-compliant Issuer and hence make all institutions connected to SURFconext capable of engaging with our SSI ecosystem. This allows end users to load institutional credentials into their wallet.
- **Interaction with Virtual Organisation (VO) Membership Management Service**
Due to a lack of time, we did not engage with a real VO membership management service like SRAM. In our demo, however, we did showcase how claims from multiple authoritative sources may be collected from issuers, and then requested by a Verifier.
During this flow, the user was in direct control of the collection and release of the claims. This process mimics closely the functional flow of combining and releasing attributes in a SRAM like system.
- **Create a Curriculum Vitae**
While we did not create an application that could use CV data, we did identify that it was very easy to register an arbitrary data schema on the ledger. As such, we tested registering an OpenBadges/edubadges schema onto the ledger which was fairly trivial. We lacked a real authoritative database for edubadges data, but were able to issue edubadges example data onto a user's wallet. While we worked with mock data and test instances, we are convinced that it is possible to create an implementation that allows for the creation of a Curriculum Vitae based on evidence from multiple Issuers.
- **Interaction with a Service acting as a verifier**
In our demo, we showcased how a service may receive verifiable claims from a user's wallet under direct control of the end user.

We have captured the various use cases in a number of demonstration videos, based on the platform that was built during this activity:

- Issuing credentials to a wallet: <https://edu.nl/fyb7d>
- Validating credentials issued by the user from a wallet: <https://edu.nl/kr8t6>

6.2 Maturity

6.2.1 Architecture and standards

The global architecture for SSI is essentially clear. However, some details still need to be fleshed out, mainly at the level of the technical standards. The architecture is layered and prescribes what components need to be deployed at one of the architecture levels.

The relevant standards needed to implement SSI are in place. These open standards are (or soon will be) managed and maintained by a standardisation body (W3C and/or OASIS), ensuring regulated governance of the open standards. Maintenance is important since the open standards are still under active development.

As the standards were created relatively recently, they are less mature compared to standards that have been around and are used in the more traditional federation landscape (such as OpenID or SAML).

In terms of completeness and maturity, we note that Verifiable Credentials and DID are fairly complete. That said, the DID standard has some properties that allow for a very broad interpretation, possibly resulting in many diverse and customised implementations, with a risk of incompatibility. The open standard DKMS is fairly complete.

DIDauth and DIDcomm, however, still seem to be in development. The documentation of these standards shows that quite a lot of details still need to be elaborated.

6.2.2 Implementation

The implementation of SSI in its building blocks is done using a variety of development languages. It feels as though all parts are written in a different language, which are then 'glued' to the other parts. While reuse of components is generally commendable, the intermix does result in a steep learning curve and a complex and an ecosystem which is potentially difficult to maintain. It was also found that libraries do not yet cover the full specification of the open standard, so some shortcuts had to be taken to implement our use cases.

Dependencies between components are high. This is mainly because Hyperledger/Indy-SDK⁵¹ is a base that is needed by most components. And since different developers are working on different components, implementations tend to be self-containing by linking in other components. As most of the building blocks are in flux, this results in version dependencies. This means that in order to run a specific component, other components of an exact version need to be included. As most components are run using Docker containers, some of the version dependency and complexity is lifted by the automation of Docker. For a production environment, or even a stable pilot, these dependencies are not desirable and should be eliminated as far as possible.

6.3 Interoperability

The concept of SSI is a common understanding with a globally applicable architecture and open standards. This, however, does not mean that interoperability and portability between implementations is guaranteed.

The DID standard, which is at the core of SSI, defines what is known as the DID method. In essence, this defines how you can interact with the DID method-specific ledger and accompanying services. From a standards perspective, this is a very elegant way to abstract the technical complexities of implementing the specific DID.

However, implementations may use technically very different DID methods, for example, implementing different types of ledger technology (e.g., an Ethereum ledger or Sovrin ledger). This will require very specific technical implementations for each DID, which makes interoperability difficult and costly.

Perhaps because of this, we noted existing mobile wallets have proven not to be (very) interoperable. As each mobile wallet provider more or less operates in their own branded ecosystem and only uses the Sovrin ledger, for example, for general operations, there is as yet little motivation to create more interoperable mobile wallet implementations. Transferring credentials between some specific wallets is currently possible using an interoperability standard, but it is largely a manual and technical process. In addition, due to the speed with which the specifications and technology move along, the actual implementation of DIDcomm for specific

⁵¹ <https://github.com/hyperledger/indy-sdk>

wallets is not always compatible with bleeding edge Issuer/Verifier implementations, causing confusing messages in the wallet interface.

A lack of available, open implementations of (mobile) wallet makes interoperability an issue too. A risk of the limited number of wallet implementations, together with the lack of interoperability, is that when the wallet supplier stops providing services, there will be a serious continuity problem for end users. Sovrin is actively working on interoperability⁵², for instance, in the field of portability of data in the mobile wallet. There are three wallets that implement the indy-sdk export and import function in roughly the same way. This allows you to save your credentials (offline) as a backup or export them and import them into another wallet.

Current wallet implementations are relatively simple and, in our understanding, will not be able to cope with a multitude of connections and credentials. This is fine for a demonstration where you have five connections and ten credentials (already resulting in 50 entries), but not for real-world situations where you need to be able to easily navigate through a wide array of credentials supplied to you by many different organisations, each organisation having one or more active connections to your wallet.

Because of the complexity of the technology used, several intermediate implementation frameworks exist to make the development of end products easier. This, however, leads to a situation where the possibilities of the end product are restricted by the use cases currently available in the framework. Specifically, Aries-CloudAgent-Python (ACA-Py) provides a restricted set of interfaces that require the developer to operate within this framework and does not allow interaction with a ledger or wallets using other means. This severely limits its usability. Similar issues arise for wallet implementations, resulting in a situation where the framework implementation becomes the de facto standard for a given technology and development based on the framework likewise determines the usability of new features, as opposed to following a centrally determined architecture and protocol.

6.4 Scalability

We assessed the scalability of the blockchain-based SSI as (very) good. The use of the DID solution contributes greatly to this scalability as a result of the decentralised setup. However, care must be taken that the flexibility of the DID can also lead to complexity and customisation. For example, according to the standard, it is possible for an Issuer to register its own DID method (see also 6.2 Interop). This means that there could be a proliferation of DID methods that would also need to be supported by all the Verifiers if it is to be able to serve the user with credentials from a "DID method silo" created by this. The freedom of the DID method therefore poses a threat to the scalability of the solution in practice.

The performance of the overall system (both throughput of the number of transactions and any latency) under large-scale use is still unknown to us. It is true that with the Sovrin implementation, only a limited number of transactions are written to the ledger. However, the core is a distributed ledger with a number of actors who play a role in a transaction and could contain millions of records. In our experiments, we found that it is relatively easy to add an extra node to a ledger, which positively influences scalability. More information about this may be found within or obtained from the Sovrin project. It is expected that this will not raise issues affecting the performance of the total system.

What we have noticed is that the development tooling and applications are currently not yet designed for scalability. The maintainability of the system also requires attention. This is due to the dependencies between projects and components, a multitude of programming languages and lack of documentation.

⁵² <https://sovrin.org/sovrin-stewards-wallet-portability/>

6.5 Privacy

It is our impression that, in general, the Sovrin/Hyperledger solution is implemented with an extremely high regard for privacy, to the point where seemingly trivial problems are countered with complicated cryptography and distributed, anonymising technologies. However, during this project, two minor specific concerns were encountered in the available wallet implementations.

When making a connection between Issuer/Verifier and Holder, the connection message allows the inclusion of an application logo in the form of a URL. This logo is displayed to the end user by the wallet when creating the connection and while displaying the list of existing connections. It is assumed that the wallet caches this logo and does not retrieve it again after the first connection. However, even this initial logo download can contain a tracking ID that will allow the Issuer/Verifier to profile the wallet application based on the HTTP headers. One would expect the wallet to regularly refresh the logo to cater for any logo change on the part of the Issuer/Verifier, or when local cache data is lost, in which case the URL would be hit again. This would provide some insight into the wallet usage, but the traceability is limited.

The second issue concerns the wallet usage with respect to the wallet app provider. Users with a wallet are compelled to use a cloud agent connected to that wallet. Currently, each wallet has its own provider specific cloud agent. This causes a situation where the wallet supplier has insight in the Holder's activities by monitoring the messages arriving at the common cloud agent endpoint. However, if each user could create his/her own personal endpoint, Issuer/Verifiers could link activities based on the endpoint to which they communicate. Either solution is a problem for absolute privacy. The suggestion in the field is that for a common cloud agent linked to the wallet supplier, the users of that cloud agent will benefit from 'anonymity in the herd' with respect to Issuers and Verifiers. Because each message is cryptographically sealed, the information disclosed to the wallet provider is minimal.

6.6 Revocation

Sovrin and the Hyperledger initiative go to great lengths to introduce a way to revoke credentials issued to otherwise anonymous, unconnected Holders. No active connection to the Holder is required (though a heads-up 'credential revoked' message is available).

In essence, the revocation implementation revolves around creating a very big number (the *accumulator*) by multiplying a long list of big numbers (the *tails file*) and applying a one-way function (a modulo or hash). The *accumulator* is published on the ledger and serves as a means to timestamp the moment until which or after which a given credential is considered valid.

Each credential receives a signed index number from the *tails file* and calculates a number that contains all the other credentials except its own (the *witness*). The Holder must then prove that it knows a number (or the index of a number in the *tails file*) and a *witness* that together create the accumulator without revealing either. This is done using Camenisch-Lysyanskaya blind signatures and elliptic curve cryptography.

Each time a credential is revoked, the Issuer publishes a new *accumulator* and a *witness delta* on the ledger. Each Holder can use the *witness delta* to modify its own *witness* to arrive at the new *accumulator*.

This implementation comes with a number of benefits:

- Revocation is largely anonymous and unlinkable: verifiers cannot fingerprint Holders based on the revocation proof and Issuers cannot determine the timing of proof requests. Verifiers will know which specific revocation registry the credential is linked to, which discloses some information depending on the size of that registry (how many others are in that registry).

- Revocation is (almost) instantaneous: within the timeframe of a single transaction.

However, we also note a number of challenges, including:

- Each revocation requires a transaction. These can be batched. But for credentials that require immediate revocation, the batch period may be short. If many credentials need to be revoked, this can lead to a lot of transactions.
- Verifiers must test against the accumulator of an issuer each and every time they process a transaction.
- Revocation proof verification requires an active internet connection and a copy of the large tails file. There is no way of checking the validity off-ledger or offline.

As an alternative to the above revocation mechanism, we note that it would be possible to rely on issuing *short-lived credentials*. As credentials may be pushed to active subscriptions, it is technically trivial to automatically reissue a new credential within the grace period of the preceding credential. This approach introduces a novel capability to the identity ecosystem which is currently not available in any of the existing protocols. Benefits of this approach would include:

- Revocation is anonymous and unlinkable, because it is contained completely in the anonymous and unlinkable credential proof.
- Proof of non-revocation does not require an internet connection.
- Revocation is automatic and instantaneous at the end of the TTL.

Such an approach would have a number of potential risks, including:

- Credentials cannot be revoked during the active period: they remain valid for the entire lifetime.
- (Automatic) updating of a credential requires an active connection to the Issuer.
- Implementation requires additional code to interpret the attributes, although this is minimal.
- A revocation status cannot be checked automatically by the wallet implementation of the Holder, but the Holder can check it visually on the credential (if the end period is a human-readable value).

Situations where this type of revocation may be implemented include, for example, prepaid subscriptions: the payment determines the length of the subscription period and also infers a non-revocable status: you paid for the period, so you should have access. For such subscriptions, a new credential can automatically be pushed to active connections that fulfil their payment obligations in time, at the time of payment.

Another revocation method involves using a non-anonymous method implemented by the Issuer. The Issuer can easily supply an additional attribute with the credential to allow the Verifier to check an Issuer-supplied API for validity of the credential: 'is this number still valid?'

This methodology will work well where the credential that needs to be validated is semi-public.

Pros:

- Revocation is instantaneous: the Issuer determines exactly when a credential is valid.
- No ledger connection required, no additional transactions needed.
- No complicated cryptography using large numbers and sizeable files.

Cons:

- The credential is not anonymous: the Verifier can link two proofs easily by using this number, thereby uniquely identifying the Holder. This can be slightly mitigated by re-issuing the credential with a new revocation number, but that creates additional complexity.
- The revocation state is not anonymous: Verifiers can automatically keep checking the validity of the credential after the initial proof.
- An active connection to the Issuer API is required.
- The revocation API is Issuer-specific and requires additional technical modifications on the part of the Verifier. Revocation status cannot be checked by the Holder using off-the-shelf wallet implementations.

Situations where this type of revocation may be implemented are where each proof already uniquely identifies a user anyway (for example: when presenting a unique registration number) and when revocation status checks can be restricted to a small set of pre-registered Verifiers, or the revocation status does not need to be anonymous (for example, for registered companies).

6.7 Trust framework

The Sovrin/Hyperledger ecosystem contains two orthogonal trust frameworks. The first one is the ledger itself and the trust one places in the transactions and parties involved directly with the ledger. The second one is the Issuer-Holder-Verifier axis that is more or less independent of the ledger implementation. In fact, the Hyperledger/Aries project, which implements the Issuer-Holder-Verifier messaging system, is trying hard to create abstract interfaces and separate the ledger interface into an independent plugin.

6.7.1 Ledger ecosystem

Trust in the ledger ecosystem is partially determined by the trust you can place in the various roles of entities interacting with the ledger. For the Sovrin ecosystem, this means that the top-level entity, the *Trustee*, has most power, followed by the *Steward* and the *Endorser*. Compromise of keys at any of these levels can mean that existing DIDs can be overwritten (or actually: overruled) or that roles of entities can be promoted or demoted at will. This specifically does not, however, give anyone the authority to remove existing transactions. In order to do that, the attacker must convince *all* nodes to drop certain transactions. As there is no automated process for that, this would involve human-human interaction with node maintainers, and hence would scale very poorly.

The essence of the trust framework around the ledger is: *do not trust what is on the ledger just because it is on the ledger*. Although the transactions are limited to specific roles and there is some sort of trust framework in assigning these roles, it is almost impossible to be certain that all public keys in the entire ledger are uncompromised. Even worse, if a key is ever compromised and the ledger is tainted with a bad transaction, it cannot be removed. It can be revoked or overruled by adding a new transaction, but the transaction will always be visible and available. You cannot expect the entire ledger to be dumped and restarted upon a single key compromise.

So the assumption must be: *trust what is on the ledger because you trust the key that wrote it*. That means there is a requirement for an out-of-band trust framework to determine the amount of trust in a specific DID or public key. The public ledger assists in tracking the history of the specific transaction and the endorsing or author key, allowing implementations to technically rule out compromised keys using simple key filtering. But effectively, an application inspecting the ledger is only going to look at a specific transaction and the specific keys involving that transaction. It is operating in a key-centric bubble of sorts, containing all the key-related schemas, credentials, DIDs and revocation registries. The function of the ledger in this respect is to supply a means to anonymously and irrevocably publish this specific bubble (e.g. schema, credential definition and issuer DID). Whatever else is on the ledger is of little concern.



Because of the inherent lack of trust in what is on the ledger, there are few risks in the keys associated with it. There is no actual content on the ledger, it is just a public ledger containing public keys with limited associated data. Anyone can write to it, but the volume of transactions is limited by the enforcement of transaction fees. On the Sovrin ledger, whoever can pay, can write to it. It's not a matter of trust, but of credits.

6.7.2 Messaging ecosystem

The additional trust required for working with privacy-related content is effectively created using the messaging ecosystem. It is set up in such a way that there are always two simultaneous and distinct channels (user's browser and a back-end cloud agent channel to a wallet implementation, possibly on a mobile device). This supplies a constant check on 'something the user knows' (browser session information) and 'something the user owns' (a separate wallet implementation).

In this framework, the Holder can visually match the Issuer/Verifier information in the browser and have the wallet cryptographically check the various requests and offerings. If the visual match is compromised (e.g., the website is redirected or compromised), the worst that can happen is that the user presents their credentials to an untrusted party. However, because of the required link to the ledger, the holder of the keys associated with the request from this untrusted party can be tracked.

What this framework does not combat is compromised Issuer sites that ask the user to present some additional proof (a SAML login, a password) before they issue a credential. This falls out of the scope of the implementation and the associated risk is at the same level as that of regular well-known sites that require a user login. It's exactly this kind of application that is required to jumpstart and insert new credentials into the entire ecosystem. The trust one can put into the framework then logically depends fully on the trust one can put into the applications inserting the data into the framework. For instance, you can trust a passport credential only as far as you trust the issuer, much like you can trust a digital SSL certificate only as far as you trust the certificate authority.

It is conceivable that a Verifier will need to maintain a specific list of credential definitions it can trust. It would then ask a Holder for a driving licence from, for example, 'the Netherlands, Germany, Belgium, France - but not before 2017 -, the UK - but not after 2020 -'. As this seems complicated, a need for an intermediate credential assembly emerges. For example, a means to validate your foreign driving licence using the Issuer of the Dutch government. A Verifier can then ask a Holder to supply a credential of this last Issuer only and redirect the Holder to that Issuer for intermediate validation. This will concentrate the trust issues for widely used credentials in specific applications.

6.7.3 Genesis file

As described in chapter 5.7, we discovered that the Genesis file contains the configuration of the ledger as it was compiled on first launch. If the configuration changes over time, even right after the first start, the genesis file will not be updated. If additional nodes are added to the configuration over time and the original nodes are removed from the configuration, the genesis file will continue to reference these initial nodes which no longer exist. This begs the question of how the general ledger should survive these kinds of configuration changes over time. From our current knowledge and experience with the Trinsic Wallet, this wallet will attempt to connect to the IPv4 and node-port addresses listed in the ledger genesis file. These IPv4 addresses must therefore be preserved over time.

Clearly, the requirement to have the original IPv4 addresses always available makes large-scale deployment and maintenance of the ledger significantly harder.

6.7.4 Wallets

The implementation of a functional wallet application on mobile devices is not a trivial task. In order to proceed with the project in a timely fashion, off-the-shelf implementations were used to interact with. These implementations adhere to the standards used by the other tools and libraries, and present a way to easily aggregate and verify credentials. During the pilot, this also led to situations where the main user interface had to be considered a black box: messages are sent in and replies received, but what actually happens inside the wallet remains a mystery. In some situations, the wallet application does not correctly parse messages, or presents an opaque 'An error was encountered' message. There is no way to look into this part of the framework, as all content is cryptographically encoded.

An opensource reference implementation⁵³ exists, but no effort was made to create a working application based on this due to time limitations. It is unknown what the actual state of this reference implementation is, with the last commit made over 9 months ago. There seems to be a similarity between various wallet implementations (especially Lissi and eSatus), which may be caused by both of these having a common root implementation, possibly linked to the reference implementation.

While working with the off-the-shelf wallets (Trinsic, Lissi, eSatus), it was quickly determined that all the implementations lack features to easily manage larger sets of connections and credentials. A user will regularly create a large number of connections to the same site. Each connection to an Issuer or Holder is named after that application, but there is no easy way to separate old and new connections of the same Issuer or Holder. Instead, a long list of connections is created through which the user has to manually search for a valid one.

The project includes a way to re-enable an old connection by having the user send a link code manually through a previously established connection. However, only the Trinsic Wallet seems to support this kind of message sending. This begs the question of how the user is supposed to interact with the connection list from a user interface point of view in other wallets. It seems likely the user is expected to recreate a completely new connection to an Issuer or Holder on every visit and therefore never look at older connections.

On the other hand, as long as the connection is available to both the Issuer and the Holder, the Issuer can automatically push new or updated credentials. This feature has a wide array of possible applications for maintaining a customer relationship. In that respect, Issuers have an interest in making sure a connection is not removed by the user and, preferably, reused on later website visits. How this would work in practice is unclear given the current wallet and protocol implementations.

Another obvious gap in all wallet implementations was a way to easily manage credentials. If SSI becomes widespread, users will quickly have to wade through a large quantity of important and less important credentials. The wallets list credentials in a huge list with no means of ordering or sorting. This works fine in limited settings for proof-of-concept projects, but for any meaningful application this will not do.

All wallets tested offered the possibility to change the active ledger to an existing well-known ledger or to a manually imported ledger. Importing a new ledger is done by manually importing a genesis file, although it seems much easier (and no less secure) to allow scanning a QR code to download that file automatically.

When changing the ledger, the wallets all indicated that you might lose existing messages. This seems to indicate that the basic wallet implementation is not able to work with several ledgers at the same time. This is rather limiting and, as far as we are aware, no technical reason exists why the wallet app should not be able to check DIDs or credentials on different ledgers. In order to be widely supported, users should not be restricted to a single ledger implementation. Supporting different ledgers with different technical implementations

⁵³ <https://github.com/hyperledger/aries-mobileagent-xamarin>

(Hyperledger, Ethereum, Bitcoin) might prove more difficult, but the DID resolving specification specifically allows such a cross-technology implementation. That is, however, only part of the challenge, as is described in chapter 6.6.5.

Creating backups of credentials on some cryptographically locked cloud storage was not possible. Every wallet has a link to a specific implementation cloud agent, but that only serves to pass on messages to and from remote applications. These agents are not able to store your credentials in some persistent way. Such an option will create a whole new range of problems regarding identification and privacy, so it is conceivable that current wallet implementations simply disregarded this feature for the moment. They do, however, offer a way to manually export all credentials to an encrypted document. This document can even be imported into another wallet of another supplier using a common passphrase construction algorithm. This requires a lot of manual steps, but does provide a very basic way to safeguard credentials collected by end users against loss or theft.

At the time of writing, each wallet app seems to be able to communicate with exactly one cloud agent, linked to the wallet upon application installation and provided for by the wallet application creator. If, however, the wallet and messaging system are viewed more like an e-mail program, one would expect a single wallet to be able to support more than one account. In other words, it would be able to connect to several different cloud agent implementations. By simply asking which agent you want to connect to when creating a new connection, the wallet could then manage several cloud agents, even supporting different ledger technologies seamlessly.

6.7.5 Interoperability

The DID specification is very broadly defined, and has the potential to serve many use cases. That may, however, also be a weak point when it comes to interoperability. As part of the DID specification, the DID method describes how entities must interact with a specific verifiable data registry. At the time of writing, over 40 DID methods exist⁵⁴, and there is no formal way of registering a DID method. Similarly, no technical, legal or trademark constraints exist in establishing a DID method.

On top of that, there are no guarantees that verifiable data registries have interoperable APIs, or have the inner workings, such as the cryptographic algorithms, been abstracted. When, for example, you compare a Sovrin-based blockchain ledger with an Ethereum-based ledger, they behave very differently. However, you are free to implement a verifiable data registry in any way you see fit. The DID:web method⁵⁵, for example, is basically just a distributed collection of URLs on top of a public web server.

The many different DID method implementations potentially mean the issuers, verifiers and wallet developers will have to support a vast array of highly diverse technical interactions within their products. In addition, as the SSI field is rather young, many of these DID methods, and specifically their technical inner workings, are still in development.

All in all, this makes it very unlikely that entities will implement many different DID methods. More likely, they will choose to implement those which either already have a large network, or will implement whatever is required because of some external requirement or dependency, such as the implementation selected by a national government.

However, this would be a direct challenge to the fundamental principles of SSI. If users, issuers and verifiers can only choose from very few implementations, this will severely limit their ability to freely engage with the SSI ecosystem as a whole. These limitations will rapidly lead to data silos where either specific data can only be

⁵⁴ <https://w3c.github.io/did-spec-registries/#did-methods>

⁵⁵ <https://w3c-ccg.github.io/did-method-web/>

used in specific contexts, or to a scenario where a select number of large players will dominate the ecosystem, similar to the situation we now see with big tech.

Recently, the interoperability dilemma has gained more attention in the SSI community^{56 57}, but this has not yet led to concrete implementations.

6.7.6 Use of the Sovrin/Hyperledger

Major technical progress has been made in recent years in the field of distributed ledger technology (DLT). DLT is about decentralised ledgers that are managed digitally. They are actually databases that are kept across a large number of computers, and these computers then exchange transactions in the databases over the network. The information that is exchanged is validated against a consensus mechanism and stored in blocks, with each participating computer keeping a full copy of the blockchain.

The blockchain itself is an application of DLT technology: it is a decentralised virtual ledger of transactions. Transactions can be registered and processed by systems without intermediaries playing a role. As a result of its decentralised nature, DLT and blockchain technology make it possible to share information and conduct transactions with a high degree of confidentiality and security.

However, not all blockchains are the same: different implementations can be distinguished. Each type fulfils a specific purpose and has a specific *raison d'être*. The blockchain implementations can be divided into three groups:

1. Private blockchains
2. Public blockchains
3. Federated blockchains (also called consortium blockchains).

Furthermore, there are various mixed forms such as public / private consent or public / private consent-less blockchains. In our implementation, we used Sovrin/Hyperledger, which is a private, permissioned blockchain.

Private and public blockchains are both distributed peer-to-peer networks, and each participant keeps a copy of the common registry. Whether the blockchain is public or private can be determined as follows based on the following questions:

1. Who can participate in the network?
2. Who can validate blockchain submissions under the consensus mechanism?
3. Who may keep the decentralised register?

The Sovrin/Hyperledger protocol is based on a private permissioned blockchain. This means that this private blockchain is managed by an administrator and is only available to a specific group of permitted participants. These participants will receive certain privileges to the blockchain, which they in turn can delegate, if necessary. The administrator therefore has an important function: he or she grants or denies participants access to the network. Furthermore, the administrator can grant or deny certain privileges to participants, e.g., write or read access. Currently, anyone who would like to use the Sovrin/Hyperledger network can participate by paying the fees. In other words, the ledger seems to be open to anyone.

The participants cannot perform validations themselves: this is done either by the administrator or by a certain group of participants according to defined validation rules. A private blockchain such as Sovrin/Hyperledger is therefore not formally decentralised and is rather a distributed ledger secured by cryptography. The advantage

⁵⁶ <https://medium.com/@decentralized.identity/the-did-dilemma-when-is-an-identifier-decentralized-9c2c68433b67>

⁵⁷ <https://medium.com/51nodes/decentralized-identifiers-based-interoperability-architecture-71afb52b4e7f>

of such a solution is fast processing of transactions. Private blockchains can process thousands of transactions per second because only a few selected participants (nodes) need to validate them. This is an advantage for long-term and widespread use.

A downside of the use of Sovrin/Hyperledger might be the limited influence that the users of the network have on the policies as a result of participating in a private network. Participants are therefore dependent on the governance and policies that are drawn up for use by Sov. This also means that there is little or no control over which parties participate in the Sovrin foundation (see footnote for details on the governance⁵⁸).

The Sovrin Foundation charges a small fee (see table⁵⁹) for writing to the public ledger, but does not charge for the act of issuing a credential: issuing credentials on Sovrin is free. We found out that there are few transactions to the ledger that would actually cost money: the initial setup for a credential issuer would be no more than a few hundred dollars, consisting of costs for an initial registration of the DID Write (for Credential Issuers), a schema definition and some credential definitions. Costs may rise depending on the amount of credential revocations, for example, after graduation of students or when temporarily pausing studies. However, costs may change in future. If the setup for the SURF SSI is based on the Sovrin/Hyperledger ledger, this could be a risk. Running a local SURF ledger would solve this.

6.7.7 Activity Flows

In the proof of concept created in this project, readily available applications and libraries were used to implement wallet and cloud agent functionality. These applications functioned more or less as black boxes in the overall flow. The cryptography and architecture of the applications make it very difficult to create a workflow that would trigger errors. Specifically, it is close to impossible to trigger the system into sending cryptographically incorrect data, or intervene with the flow of messages. Once the activity flow between Issuer-Holder and Holder-Verifier is made possible, the technology is geared completely towards 'happy flow', offering very few handles which can be used to make changes to the process. This makes it much harder to develop custom workflows.

For example, an initial version of the PoC application implemented the messaging architecture itself, but this caused problems creating a working activity flow. Due to time constraints, it was decided to test the ACA-Py cloud agent as an alternative to this custom implementation, and this seemed to work out of the box. However, this introduced ACA-Py where the whole messaging interaction is a black box. It must be noted that ACA-Py gives ample options to plug in or override behaviours by adding external code, and examples of these are provided too.

⁵⁸ <https://sovrin.org/library/sovrin-governance-framework/>

⁵⁹ <https://sovrin.org/issue-credentials/>

7 Follow-up

Now that the project has come to an end, the knowledge and experience gained can be used to conduct a much more focused discussion about the potential value and the possibilities and impossibilities of SSI. Based on this report, internally oriented knowledge dissemination is a logical next step. The described applications (use cases) can be demonstrated, making it easier to conduct discussions with business owners within SURF. Initially, edubadges, eduID and SRAM would appear to be the most obvious contenders. These conversations may provide new, inspiring examples of possible uses of and requirements for SSI. It will then be possible to explore whether these applications can be supported in the form of a proof of concept where the current demo environment would be extended to leverage these platforms for testing real data exchange.

The experiment that SURF carried out within this project was deliberately small in scale and internally oriented. The goal was to gain a feel for the technology stack and to set up an implementation with it. This approach has taught us a lot about both the standards and the technology. Most of our findings are based on implementing a few happy flows of the use cases around which this SSI project revolves. The technical details of these flows are known to a large extent, but it is recommended to specifically determine and test the following non-happy flows:

- Presenting invalid credentials
- Presenting revoked credentials
- Presenting not-yet-unrevoked credentials
- Mitigating traceable Logo URLs
- Mitigating - if possible - traceable cloud agent API endpoints

Furthermore, the current wallet implementations do not give much feedback about what is actually happening internally. Having a transparent wallet implementation would provide a lot of insight into what is happening under the hood.

The current proof of concept environment consists of various separate instances of components that fit together, but do not yet provide a solid, scalable environment. The next step would therefore be to expand SURF's SSI implementation into an environment which can be used to conduct real experiments. It is conceivable to use current knowledge to build a lab environment in which the SSI implementation will also be accessible to Dutch educational and research institutions that want to gain experience with SSI. A side effect of this will be that knowledge and experience could be gained with the effects of a larger-scale application of SSI, including scalability, performance, manageability.

During our investigation, we noted that specific ledgers have different properties, for example, the ability to create "smart contracts" within Ethereum. We have not tested these capabilities at all, as we saw no need in the context of our use cases. However, that does not mean such use cases may not exist within our community. It is therefore recommended that multiple ledger types should be supported in any SSI lab. This would allow the testing of various use cases and also provide a practical environment to compare various approaches.

In summary the following steps are suggested:

- Internal knowledge dissemination about SSI.
- Discuss with business owners within SURF, starting with edubadges, eduID and SRAM.
- Identify critical benefits, if any, in the aforementioned services that SSI-based technology would bring. This will improve understanding of the business value for engaging with SSI technology.



- Identity functional and architectural components of the SSI architecture that could create value within our use cases. Determine which aspects are less important and might be operated in a centralised way, for example, without running a full-blown distributed SSI infrastructure.
- Specifically determine and test possible 'non-happy flows'.
- Use current knowledge to build a more stable lab environment.
- Make the lab environment available to educational and research institutions.

8 Conclusions

The deep dive into SSI has been a very interesting experience. From a functional perspective, Self Sovereign Identity is both fascinating and challenging at the same time.

SSI tries to utilise a complex technical solution to guarantee absolute privacy and anonymity. While deploying and testing the Hyperledger-based BCGov platform, we noted that many of the core aspects of SSI are indeed implemented to what seems to be a sufficient level: anonymity is preserved throughout the ecosystem and linkability is prevented. Interoperability seems to be sufficient, as we were able to combine components provided by multiple vendors into a usable and working technical implementation.

At the same time, the SSI ambition is to give end users direct and unprecedented control over the dealings with their personal data. Combining this with the technical complexity of the platforms poses a number of challenges. Especially in the field of key revocation and wallet recovery, we have the impression that current solutions might be challenging for end users. We wonder whether enough of the technological challenges could be resolved to make this solution intelligible to and usable by an average end user.

We were able to successfully test the major usage patterns for use cases we collected, so we are confident that SSI can fulfil the functional requirements, albeit using mock data in most cases. Users do get the freedom to use the attributes they have collected as they see fit, and it is clearly possible to build a trusted platform based on a distributed ledger, if the technology layer is complemented with additional policies to regulate participants.

When looking at the technical side, the current state of SSI presents a very mixed environment, which still shows many signs of being in its early days. The sheer number of start-ups, pilots and trials makes it very hard to pick out what is really working and what the state of development is. While several aspects of SSI are very well architected and implemented, others seem to be lacking, even at the standards level, forcing implementers to make ad hoc choices. In several cases, we noted that this has a negative impact on security, interoperability and usability.

As we have showcased, implementing a stand-alone environment which would operate within a limited scope, for example, within the Dutch higher education sector, is clearly already possible. However, attention would be needed for operational and scalability aspects, as well as usability. In regard to the latter, it seems inevitable that a wallet app would have to be developed to allow for much better control over the end user interface and the wallet capabilities. To maximise the possibility of reuse of libraries and components, it is recommended to set up such a platform based on a technology that already has sufficient traction, e.g., Hyperledger.

For many of the public and commercial platforms currently available, it is not yet clear how either governance or sustainability will develop. Any cloud or as-a-service-based platform choice should therefore consider an exit strategy right from the start. An additional strategy could be to start with use cases where there is no need for very long persistence of the information on the ledger. This could allow for testing with real-world use cases without the need for long-term commitments to maintain a ledger infrastructure.

For SURF and the institutions, with the well-established SURFconext federated AAI, replacing the current authentication infrastructure would add little direct value and would cause huge disruption.

For scenarios in which organisations in the Dutch higher education landscape interact with other parties, it would be wise to take a number of trends into account:

- **Direct control over personal data (“Regie op Gegevens”)**
A number of governmental and quasi-governmental initiatives exist with the aim of decentralising the

exchange of personal data and allowing citizens to be in direct control of the exchange. Notable examples in healthcare are medMIJ⁶⁰ and NUTS⁶¹, and the recently published digital identity vision memo (“Visiebrief digitale identiteit”⁶²), announcing the introduction of a DBI (Dutch Base Identity) by the Dutch government.

While none of these directly reference distributed identity as a technical solution, it is clear that SSI-based solutions are both well-suited and already on the radar as potential mechanisms to enable the proposed transformation in these sectors.

It is likely that Dutch institutions will be expected to interact with these platforms either as Issuers or Verifier. It would therefore be wise to gain experience within SURF with the interaction with such environments.

Furthermore, if Self Sovereign control becomes the dominant model for exchanging personal data, we may expect strong pressure on SURF and the institutions to facilitate data exchange in this way, alongside existing mechanisms.

- **Reducing complexity**

While it may seem contradictory, given the complexity of the underlying technology of the concept of SSI, SSI could be used to reduce complexity at other levels.

This is due to the fact that SSI allows for the decoupling of relations between various entities in the identity ecosystem. With the use of SSI, it is possible to:

- Allow more direct end user control. By some, this is proposed as a way out of several GDPR complexities, as you could consider this to be equivalent to “consent”.
- Decouple authentication from attribute release, allowing for more flexibility in the use of attributes. This could allow users to leverage attributes in scenarios where there is no contractual relation (or none is required) between the institution and a service.
- Aggregate claims from other multiple authoritative sources in a transparent and scalable way, currently only possible using proxies. This would also more easily allow for re-use of existing claims for other sectors and reduce the need for proxies.

Within SURF, several developments are consistent with these trends:

- eduID leverages a user-centric identity model and allows for linkage of institutional attributes to that identity. The eduID model and its goals overlap with SSI, although eduID currently deploys a proxy infrastructure.
- SRAM allows for delegation of control from the institutional level to the researchers directly involved in specific communities. Again, proxies are used as focal points, both technically as well as on a functional level.
- edubadges facilitates an ecosystem where students can more independently collect accreditation marks from institutions, and edubadges would strongly benefit from a persistent, user-centric identity model. (edubadges currently leverages eduID.)

It is likely that these developments could benefit from leveraging SSI-based capabilities and testing this assumption seems like a logical next step.

⁶⁰ <https://www.medmij.nl/>

⁶¹ <https://nuts.nl/>

⁶² https://www.tweedekamer.nl/kamerstukken/brieven_regering/detail?id=2021Z02985&did=2021D06488

It is known that several research groups in the Netherlands are involved in SSI development. However, it is currently unknown to what extent institutions are investigating the use of SSI technology. Since authentication and authorisation is typically a supporting technology for the primary processes, its replacement is probably not very high on the agenda. In some areas, however, SSI may be a key enabler, either because of its new technical and functional capabilities, or due to the ability to reduce technical or legal complexity or costs. SURF should invite and help researchers as well as institutions to collaborate on investigating such scenarios to gain experience in these areas. Not only will this help shape the SSI ecosystem in the Netherlands, but it will also help SURF to determine its role in the ecosystem.

With the introduction of the SURFconext platform in 2009, SURF has taken the role of a technical facilitator when it comes to AAI. On behalf of our institutions, SURF reduces technical and organisational complexity and has helped reduce cost at our institutions. At the same time, SURF has been at the forefront of AAI technology, security, data protection and privacy. Should SSI as a concept succeed, the transition from federated to distributed identity is an excellent opportunity for SURF to use its many years of knowledge and technical skills to assist education and research institutions in this transition.

9 Online resources

In carrying out the project, the team collected background information in a public wiki space. In addition, all code created as part of this activity has been made available under the Apache 2.0 licence. We also created two videos to demonstrate the various use case patterns based on the demonstration platform.

Background information (wiki)

- <https://wiki.surfnet.nl/display/SSI/Self+Sovereign+Identity+Home>

Source code repositories

- <https://gitlab.surf.nl/ssi/deploy-playbook>
- <https://gitlab.surf.nl/ssi/issuerverifier>

Demonstration videos

- Issuing credentials to a wallet: <https://edu.nl/fyb7d>
- Validating credentials issued by the user from a wallet: <https://edu.nl/kr8t6>

Publication details

Editors

Niels van Dijk (SURF)
Bart Kerver (SURF)
Harry Kodden (SURF)
Michiel Uitdehaag (SURF)

With thanks to

Arnout Terpstra (SURF)
Michiel Schok (SURF)

Copyright



This publication is licensed under Creative Commons Attribution 4.0 International:
<https://creativecommons.org/licenses/by/4.0/deed.en>

May 2021