Research Internship Project Report

Matti Eisenlohr
S1000794 (RU)

February 8, 2022

Contents

1 Introduction

For some time now, SURF has provided a Single Sign-On method in the form of SURFconext, which acts as an intermediary between services (e.g. OSIRIS, Brightspace, ANS) and identity providers (e.g. institutions) as part of the federated identity model.
One of the long term goals of SURF is that users do not need a service such as SURFconext as the broker for their digital identity, but that users can instead become the broker between their identity providers and services themselves. For this purpose eduID was introduced, an ID that anyone can activate for themselves. Currently, eduID is used as a method for to grant guest access to some services, thus as just another identity provider, however with limited applications. One of the more long term goals for eduID is to turn it into a proper Self Sovereign Identity (SSI), where users own and control their own identity . Examples of existing SSI concepts would be IRMA by the Privacy By Design Foundation in Nijmegen and Sovrin, that utilizes blockchain technology. Since SSI is a rather new concept for SURF (but also in general), they are still in the process of getting familiar with the technology and still have many questions and a long way to go until eduID can become a proper SSI.
An important aspect of such an SSI solution is the digital wallet, an app on a users phone where they store their credentials and from which they control what happens to those credentials.
While my assignment was only very vaguely defined in the beginning, my research was essentially to create a first concept for what such an eduID SSI wallet could look like and which standards could be used in a potential implementation. I started with writing down use cases and requirements for the wallet app. I then proceeded to create frontend flows and write down first considerations for these use cases. In that vein, I created a prototype using Figma (see `https://www.figma.com/file/S2e5VBKvAIwPj7SJTCOWKR/eduID-wallet-app`).
After that I focused on the backend aspects of the wallet app. For this I first examined the two most important data model standards in SSI, Verifiable Credentials and Decentralized Identifiers. Additionally, I also examined IRMA and its data model and protocols, but focused mainly on the other two as they are much more standardized and adopted with the SSI community. I then defined the eduID SSI ecosystem with all the roles and participants within it and discussed some data storage issues that need to be considered. And finally I researched standardized SSI protocols, of which I found two in the form of DIDComm and OpenID Connect Self-Issued OP, and some standardized ways to implement features like revocation and credential renewal.
At the very end of my internship, I wrote down my conclusions (what should SURF focus on, which standards to use) and gave recommendations on how to proceed and where more research or discussion within the eduID project team is needed.
All my work has been documented on SURF's internal eduID wiki in the form of multiple different pages to cover different aspects. This report gives a summary of what is documented on the wiki pages, sometimes even whole excerpts from a wiki page, however it will not cover everything I documented, but only the aspects I deem strictly relevant. The full details of my results remain on the wiki, where descriptions also include much more technical details.

2 Basic Concepts

In this section I will give a short description of the basic underlying concepts of this research.

2.1 Attributes/Claims

Attributes and claims are two terms that essentially mean the same thing. An attribute or a claim is essentially a statement about a specific person. Examples of a claim/attribute are birth date, first name, second name, student number, etc.. Services often want to know specific properties about users, for example to see that the user is a student. For this purpose they request users to present specific claims/attributes to them together with a verification mechanism to proof the legitimacy of this statements. Claims/attributes always have an authoritative source, called an issuer. This authoritative source is the party that is actually asserting these properties about the person. Services trust in the legitimacy of presented claims because they recognize its issuer as a party that has the authority to assert these claims.

2.2 Credentials

A credential is essentially a digital version of physical cards one would carry around in a physical wallet, like a government ID, a passport, a debit or credit card, drivers license, student ID, etc.. Such a credential consists of a collection of different claims, just like their physical equivalents do, and is usually digitally signed by its authoritative source.

2.3 Issuers

As already mentioned, issuers are the authoritative source that are asserting different claims and providing credentials containing these claims to individuals. They act as the identity provider for users within an SSI environment. For information about the role of issuers, particularly in relation to eduID, see section 5.2.

2.4 Verifiers

A verifier is a service that requests user to present certain claims to either prove certain properties about themselves (e.g. if they are a student or if they are over 18 years old) or authenticate the user. For more information about the role of the verifier, especially in the context of eduID, see section 5.3.

2.5 Digital Wallet

Akin to a physical wallet, where one usually carries around a bunch of different cards, a digital wallet is a piece of software, usually a smartphone app, where users store their digital cards, i.e. credentials. However a digital wallet does much more than just store credentials. It also facilitates interaction with other parties (i.e. issuers and verifiers) and derives so-called presentations from the credentials within the wallet. These presentation contain a selection of claims (or even just one claim), that have been requested by a verifier in a way that only the requested claims need to be presented instead of having to present entire credentials, which also contain claims that the verifier does not need to know.
The wallet is thus the component with which users interact with the wider SSI environment and is as such a very crucial part of an SSI ecosystem. For eduID, it is also important that the wallet provides the user with transparency about past interactions of the wallet with services.

2.6 eduID account

The eduID account is mainly an email address that is linked to user. This email account is used to identify the user when setting up the wallet app. This is especially relevant for the case that a user has already used the wallet before but is switching over to a new phone. In the current state of eduID, this account is essentially eduID and also contains information such as login methods, verified personal data (name, student in the Netherlands) as well as data about the services accessed via eduID. In a potential future when eduID becomes a sort of SSI, many of these functionalities will go over to the eduID wallet app, but the account with an associated email address and optionally a password or security key to act as a way to be able to identify an eduID user, particularly during the setting up of the app and the creation and loading of back-ups.

2.7 High-level SSI architecture

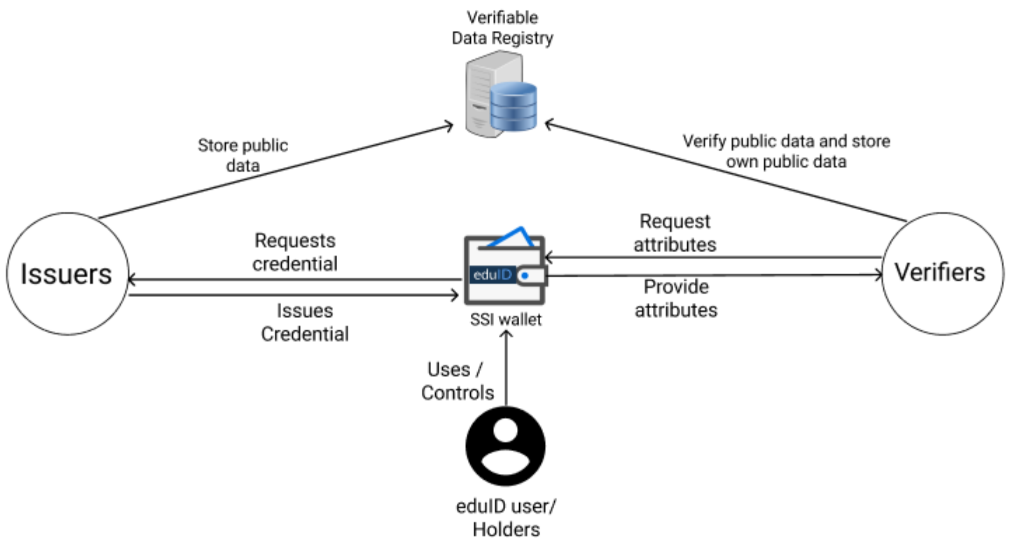


Figure 1: High-Level SSI architecture

Figure 1 highlights the basic interactions and relationships between the parties involved. This figure also includes the role of the verifiable data registry. This is not an active participant, but rather a data structure used to store data that should be publicly available to all participants in a verifiable manner. More about the role and form of this registry can be found in section 5.4.

3 Use Cases & Requirements

As the first step of my research, I defined relevant use cases for the eduID wallet app. I ended up with 12 (or 11, as two of the use cases are very similar to each other), some of them are absolutely essential, some of them are more use cases that are nice to have, if they can be implemented well, but not essential. The essential use cases I defined were:

1. Setting up the app (registration) - either with or without an existing eduID account.
2. Adding credentials to one's wallet
3. Logging into a service using credentials stored in the wallet
4. Disclosing claims/attributes from one's credentials without logging in (Use Cases 3 & 4 could be summarized as one single use case "Disclosing claims/attributes")
5. Deleting credentials from one's wallet
6. Getting an overview of all credentials in the wallet and inspecting them in detail
7. Revocation of a credential by its issuer (the authoritative source of the credential), i.e. invalidating a credential

The optional/nice-to-have use cases I defined were:

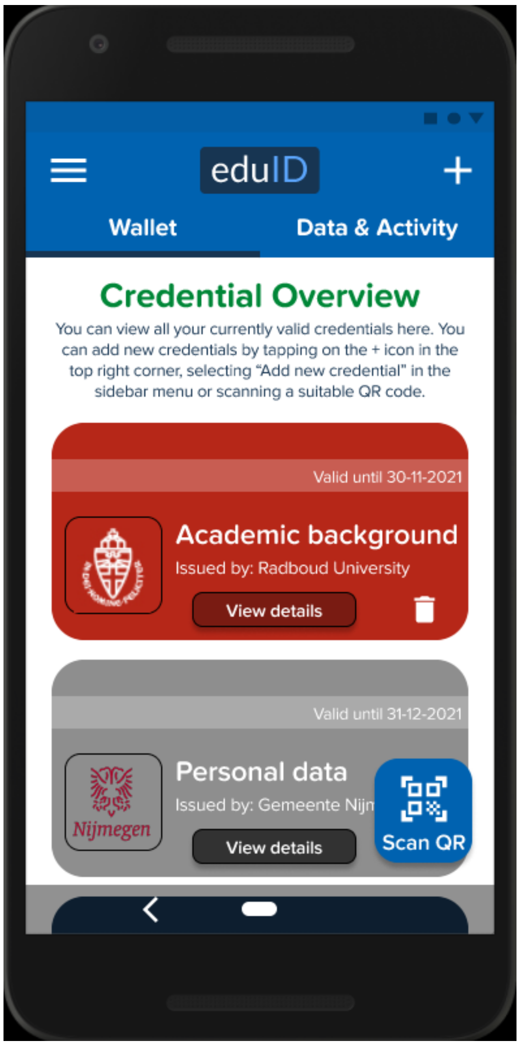
1. Correcting inaccurate credentials. One could argue that this is not a feature of the app itself, rather users need to contact the issuer to correct the underlying data, but they still need to renew their credential in the app.
2. Renewing credentials to extend their validity period. This and the previous use case both revolve around the renewal of a credential, so they could be summarized into one bigger use case "Renewing credentials". They are still functionally different, but from a pure UI-perspective within the app itself, they are somewhat similar.
3. Getting an overview of the services users have interacted with and which attributes/claims these services currently store. In the context of eduID, there are many situations in which it makes sense for a service to store certain claims about eduID users, for example when a user registers for a course, so it would be great to provide some transparency about this here. It would also be ideal if the wallet logs all past interactions and allows users to inspect these interactions (when?, which attributes/claims were disclosed?, if credentials were issued: which credential was issued?).
4. Revoking consent for said storage. Where claims are only stored based on consent, this is a GDPR requirement for these services. However it is not required for claims where another legal basis as defined by the GDPR can be invoked. It might be nice if the app could at least provide a link to some page of the service in question, where consent might be revoked to streamline this process, however it is by no means a must-have feature.
5. Creating a back-up of the eduID app data that could be used for recovery.

In addition to these use cases, I also defined some requirements, mainly relating to user friendliness in the UI and compliance with the GDPR, and some smaller functional requirements, like being able to add credentials from different sources. Two requirements by SURF are that a) the app should use open standards wherever possible and b) that users should have a different identifier for each relationship with a service, such that they cannot be (easily) tracked across multiple services. At the very end of my internship, I also added requirements relating to implementation specific aspects. Wherever relevant, I will mention them in section 6.

4 Frontend flows & Demo

The next step then was to visualize what such an eduID app could look like in practice. For this purpose I took inspiration from existing wallet apps such as IRMA or Lissi as well as the look and feel of the current `mijn.eduid.nl` website, which is where you can currently manage your eduID account, and created a prototype using Figma. At this stage I mainly focused on the front end and its user-friendliness and usability. I did not pay much attention to the backend at this stage yet.

4.1 Home screen

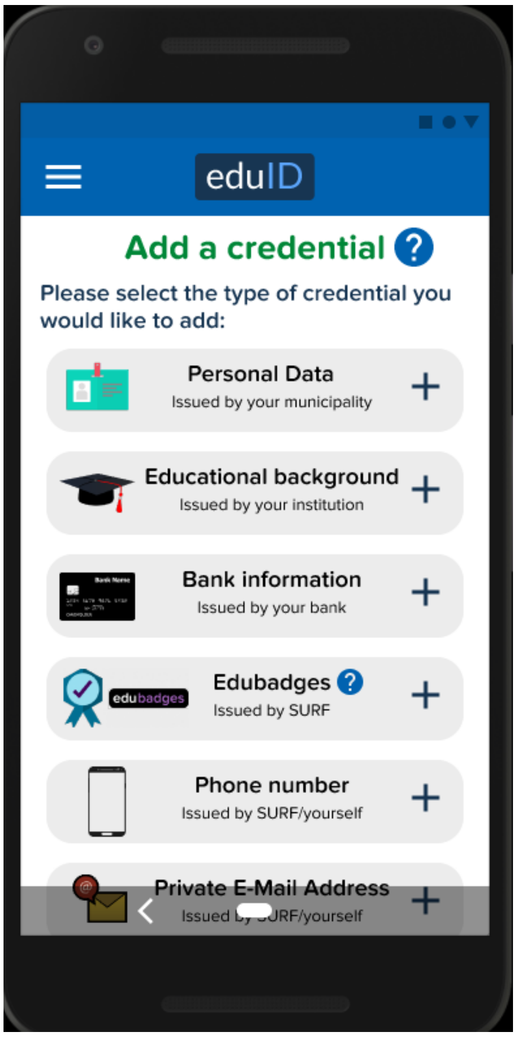


You can view all your currently valid credentials here. You can add new credentials by tapping on the + icon in the top right corner, selecting "Add new credential" in the sidebar menu or scanning a suitable QR code.

The figure on the left shows the home screen of the prototype I designed. This home screen works as an overview page from which you can access all relevant features. It provides an overview of all credentials that are currently in the wallet, which can be inspected further by tapping on the "View details" button.
The screen takes clear inspiration from most other wallet apps I found. These apps also use a credential overview page as their home screen with credentials visualized as an abstraction of physical cards like a banking card or an id card, like one would carry in an actual physical wallet.
Many apps also feature a second tab to show past connections to services with the app (called "Data & Activity" here, named after the tab of the same name on mijn.eduid.nl, which does essentially the same thing). The so-called "hamburger menu" (the menu symbol) in the top left corner opens a sidebar menu when tapped on and can be used to access other functionalities and tabs of the app.
One thing to note here is that it might make sense to change the placement of the Scan QR Button to make it stick out more from the other elements on the screen. This button could be included within the bar at the top or might be included as a bar on the bottom of the screen.

4.2 Adding new credentials

For the use case of adding new credentials to the wallet, I created two different flows. These flows differ in how the process is initiated, otherwise they are mostly similar. The first flow involves scanning a QR code to initiate the process, which is the most common method found in existing wallet apps. This works fine for the most part and is pretty straightforward. However, when trying to test out some wallet apps, I did not know where I could retrieve credentials from apart from some demos I found, which can be kinda frustrating as a user.


eduID

Add a credential

Please select the type of credential you would like to add:

Personal Data
Issued by your municipality

Educational background
Issued by your institution

Bank information
Issued by your bank

Edubadges
Issued by SURF

Phone number
Issued by SURF/yourself

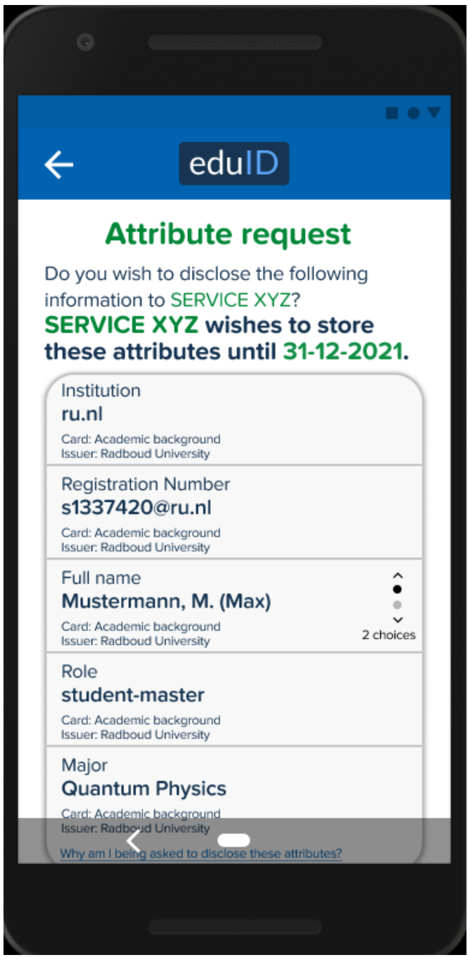
Private E-Mail Address
Issued by SURF/yourself

Figure 2: IRMA style "Add Credential" page

For this purpose I added a second flow, which is heavily inspired by IRMA. IRMA offers an extra tab, where a user can select some basic credential types to add to one's wallet, using a specific issuer and identity verification process. This is really nice because it saves users time they would otherwise need to spend searching for suitable QR codes and makes it possible to immediately add some basic credentials. QR codes are still the main way to add credentials, but having this tab as an extra should make the user experience a little bit nicer. Figure 2 shows what this screen looks like in the prototype. Tapping on the question mark symbol at the top gives a further explanation of this screen. I originally had the explanation on the screen itself, but that made the page really bloated, so I opted to hide the explanation in a pop-up window instead.
After initializing the process by scanning a QR code or selecting a credential type on the "Add new credential" tab, users will first have to identify themselves to the party that will issue the credential. This identity verification process will look different for each issuer and could for example take the form of a login (e.g. via DigiD for a government-issued credential or via SURFconext into an institutional account, like it works in IRMA) or by disclosing specific claims/attributes from credentials that are already in the wallet.
After verifying their identity, users then get presented with the credential they are about to be issued, which they can either accept so that it gets added top the wallet, or decline, in which case the credential will not be issued.

4.3 Claim/Attribute Disclosure

Disclosing certain claims to a service, regardless of the context, is fairly straightforward. It starts by the user scanning a QR code and then being provided with a request for specific claims (see figures 3 and 4). If a specific type of claim can be retrieved from multiple different credentials, then the user should be able to choose where to get this claim from. If the service intends to store claims after the session they should also include a date until which they intend to store the claims. They could also provide an explanation here to be more transparent about why they requested these claims in particular. The user can then either decline the request and not disclose these claims to the service (called the "verifier") or accept the request and disclose them to the verifier. If they accepted the request they should then be able to see the service in the Data & Activity tab of the app and see which claims they have in storage.



eduID

Attribute request

Do you wish to disclose the following information to SERVICE XYZ?

SERVICE XYZ wishes to store these attributes until 31-12-2021.

Institution
ru.nl
Card: Academic background
Issuer: Radboud University

Registration Number
s1337420@ru.nl
Card: Academic background
Issuer: Radboud University

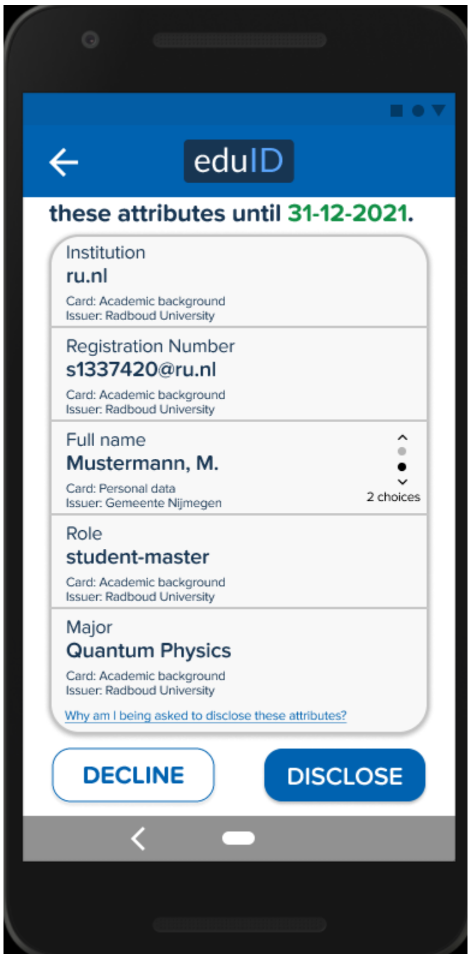
Full name
Mustermann, M. (Max)
Card: Academic background
Issuer: Radboud University
2 choices

Role
student-master
Card: Academic background
Issuer: Radboud University

Major
Quantum Physics
Card: Academic background
Issuer: Radboud University

Why am I being asked to disclose these attributes?

Figure 3: Attribute request upper part



eduID

these attributes until 31-12-2021.

Institution
ru.nl
Card: Academic background
Issuer: Radboud University

Registration Number
s1337420@ru.nl
Card: Academic background
Issuer: Radboud University

Full name
Mustermann, M.
Card: Personal data
Issuer: Gemeente Nijmegen
2 choices

Role
student-master
Card: Academic background
Issuer: Radboud University

Major
Quantum Physics
Card: Academic background
Issuer: Radboud University

Why am I being asked to disclose these attributes?

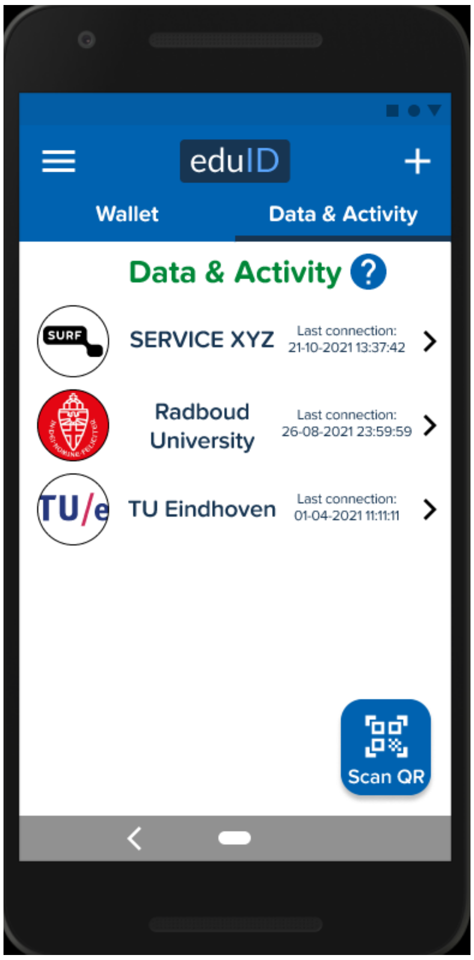
DECLINE DISCLOSE

Figure 4: Attribute request lower part

4.4 Other flows and screens

Apart from the aforementioned screens and flows, I also created an app registration flow, where users link the app to their eduID account via a one-time password and set up a PIN and optionally also a fingerprint, which they then later use to log into the app. Consequently, I also designed a login flow, where users use the aforementioned PIN or fingerprint to login.

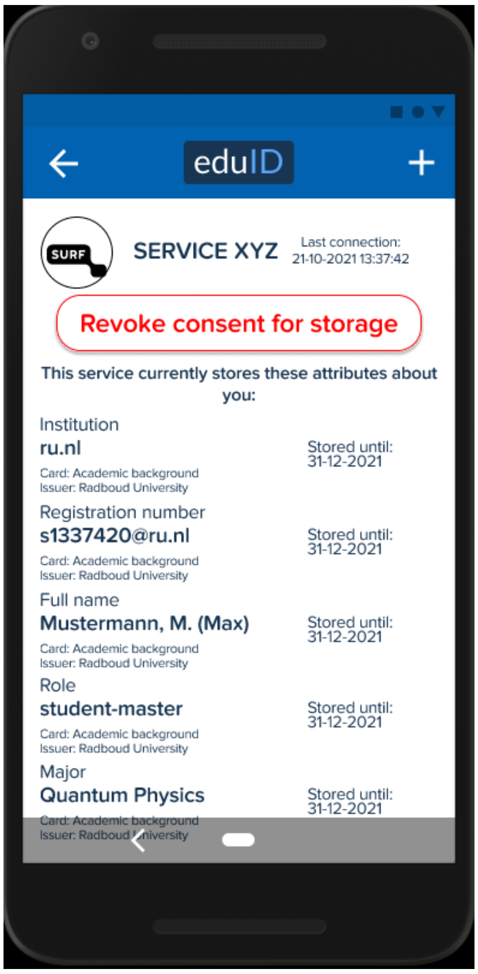
In the data and activity tab, users can see services they interacted with in the past, that still hold some of their claims/attributes. When tapping on a service, users can see which claims the service currently has in storage and until when they have them in storage. I also put a "Revoke consent" button on the screen, which could either send a message to said service to



Data & Activity

SERVICE XYZ	Last connection: 21-10-2021 13:37:42
Radboud University	Last connection: 26-08-2021 23:59:59
TU Eindhoven	Last connection: 01-04-2021 11:11:11

Figure 5: Data & Activity Screen



SERVICE XYZ Last connection: 21-10-2021 13:37:42

This service currently stores these attributes about you:

Institution ru.nl Card: Academic background Issuer: Radboud University	Stored until: 31-12-2021
Registration number s1337420@ru.nl Card: Academic background Issuer: Radboud University	Stored until: 31-12-2021
Full name Mustermann, M. (Max) Card: Academic background Issuer: Radboud University	Stored until: 31-12-2021
Role student-master Card: Academic background Issuer: Radboud University	Stored until: 31-12-2021
Major Quantum Physics Card: Academic background Issuer: Radboud University	Stored until: 31-12-2021

Figure 6: More details about Service XYZ

delete the claims or could link the user further to a website of the service where they can revoke their consent. Again, this is more of an optional feature and might be tough to implement and enforce, so this might be scrapped.
Additionally I also created an "Account" tab, which functions very similarly to the tab of the same name on the eduID website. On the tab, users see information about their eduID account and can download or delete data from their eduID account or from the wallet app as well as decouple their account from the app.
The full Figma project can be found at `https://www.figma.com/file/S2e5VBKvAIwPj7SJTCOWKR/eduID-wallet-app`. There the demo prototype can be started by clicking on the play button in the top right corner. Alternatively the demo can also be accessed via `https://www.figma.com/proto/S2e5VBKvAIwPj7SJTCOWKR/eduID-wallet-app?node-id=2%3A2&starting-point-node-id=2%3A2&show-proto-sidebar=1`. On the demo one can select the flow one wants to inspect on the lefthand side.

5 EduID SSI ecoysystem and goal architecture

The Verifiable Credential standard (see section 6.1) defines multiple roles within an SSI ecosystem. These roles need to be applied to the context of eduID. Figure 7 models the relationship between the different roles and provides a general overview of the eduID ecosystem.

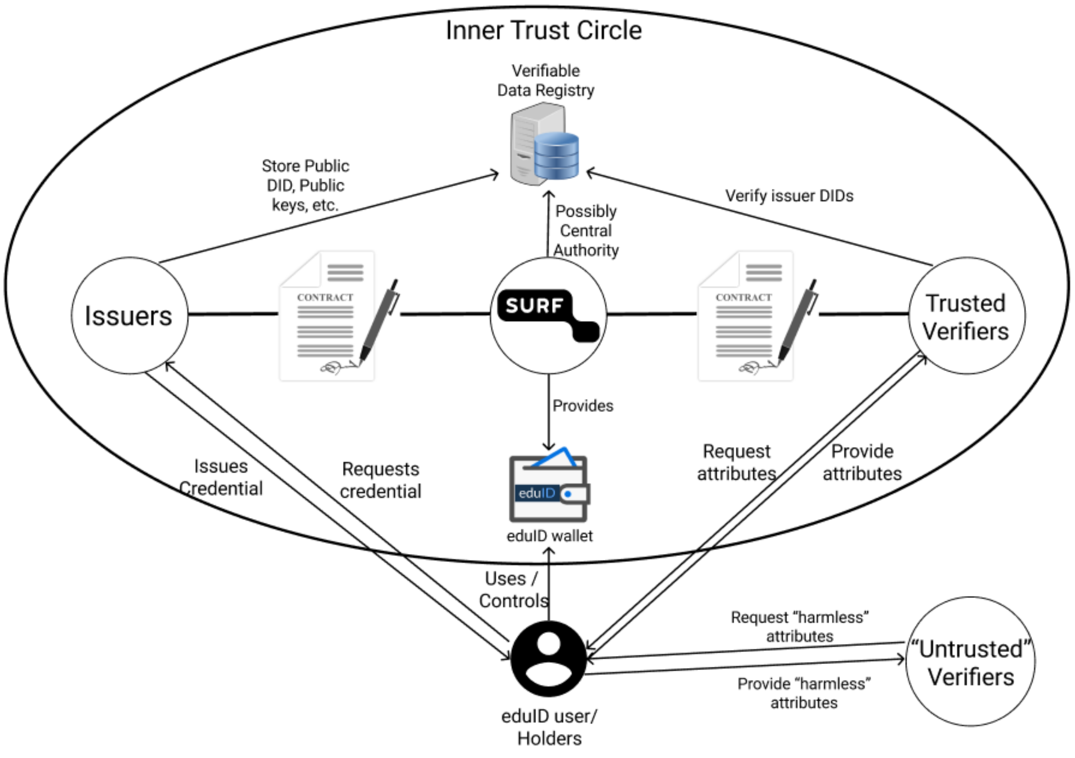


Figure 7: Overview of the eduID ecosystem

5.1 Holders and Subjects

The most basic role is that of the holder. According to the VC specification they are defined as storing one or more verifiable credentials and deriving verifiable presentations from them. This essentially means that everyone who has an eduID account and has an eduID wallet linked to it is the holder of that specific wallet and its contents. In practical terms, this role is filled by students, researchers, professors, university staff and former students (alumni) (possibly also external contractors that need access to certain university systems?).
Connected to the role of the holder is that of the subject. A subject is someone the underlying claims contained in verifiable credentials are about. Under the assumption that in eduID, users only hold credentials that actually refer to them and do not hold credentials for someone else (for which I don't really see any reason in the context of higher education), the holder and the subject are always the same person within the context of eduID, which is why I grouped those roles together here. The role of the subject is also closely connected to that of the data subject in the GDPR. In fact, from a legal point of view, all subjects are also data subjects under the GDPR.

Apart from the already stated functions, they do not have any special obligations or responsibilities other than that they should make sure a) that they only accept credentials that do not contain any inaccurate data/claims and b) that their credentials are refreshed on time, if they intend to use them beyond the predefined period of validity.

5.2 Issuers

Issuers are defined as entities that assert claims and create verifiable credentials from them and transmit ('issue') those credentials to holders. In short issuers hold some sort of personal data about eduID users and issue them credentials that prove certain claims about them based on that data. Potential issuers within the context of eduID are primarily educational institutions, but banks, municipalities/governments (e.g. via DigiD), SURF, Studielink and DUO also make good candidates to fulfill the role of issuer.
This role has many responsibilities and requires probably the most work when compared to the other roles. First of all, in order to even become an issuer, they would first need to enter the eduID trust framework by entering a contract with SURF, similarly to how issuers must sign a contract with the SIDN in IRMA. This contract should define what kind of credentials they may issue, for how long these credentials should be valid as well as which of the advanced concept they may and/or should include in their credentials. The contract will also impose some obligations and responsibilities on those issuer. Issuers are also bound by the GDPR, as they will probably be classified as data controllers in many cases, which already brings a lot of responsibilities with it.

5.3 Verifiers

The VC specification defines verifiers as entities that receive verifiable credentials and/or verifiable presentations for processing. What this means is that anyone asking a user for specific data or to prove certain properties about themselves and then processing said data becomes a verifier. This role will be filled by parties that offer services to eduID users, which means that (again) mainly educational institutions will be verifiers. However, other parties such as DUO, SURF and Studielink might also be sensible verifiers.
Who exactly can become a verifier is a bit more complicated as it depends on what scenarios eduID is intended to be used for. If it is only supposed to be used directly within the Dutch higher education sector, the pool of potential verifiers is limited and it's at least feasible to sign individual contracts between SURF and potential verifiers, even if somewhat inefficient. However if the scope is expanded such that more fringe use cases might occur, this becomes less feasible. One example of such a fringe case would be a researcher at a Dutch university wanting to access a very specific and rare paper/book (e.g. some really niche book from the 1600s) that is only available at one university somewhere abroad. That university requests the researcher to log in to verify who wants to access the paper/book and that they are indeed a researcher in that field. Normally they would need to request all this data from their institution and go through a more complicated process to prove all these properties to the requesting university. This could be simplified by using VCs via eduID. However it becomes quite unfeasible to set up individual contracts with all possible universities. A simpler example would be places like Domino's offering student discount when ordering online. When going physically to the store, this is as simple as showing one's student card s they are not gonna keep any of your data after that. But when ordering online and proving your identity online, this is not that simple as they likely want to keep the data for marketing purposes. One could use eduID to present a claim that proves one's student status. If this is an intended use case, the number of potential verifiers rises again.

Taking this into consideration, there are three possible approaches (I can think of) to determine who gets to be a verifier:

- **Open system:** Anyone can become a verifier as long as they set up the necessary infrastructure (and maybe agree to some general terms and conditions). In this scenario, the claims/attributes a verifier can actually verify can still be limited via the "Terms of Use" property in a credential, however that can only be done by issuers and would need to be actively be enforced by the wallet. Holders could also present derived claims to hide certain parts of their credentials and claims to limit the data they actually present to verifiers.
  In this case, general trust in verifiers by holders and issuers is lower than in a less open system. Here it would be important for individual users to actually pay attention to the party that is requesting certain properties from them and if they trust them enough to actually disclose these values.

- **Semi-open system:** In principle, anyone can become a verifier as long as they set up the necessary infrastructure (and maybe agree to some general terms and conditions). However they will only be able (/allowed) to verify certain claims and credential types. For some more sensitive credentials, they will need to enter a contract with SURF that specifies their rights, privileges, obligations and responsibilities in detail.
  With this approach you would gain an extra degree of trust regarding verifiers that have actually entered into such a contract. However users would need to trust their wallet software to actually enforce these policies, so the wallet needs to know which parties are in a contractual relationship with SURF and which are not.

- **Closed system**: Like issuers, verifiers need to enter a contract with SURF to become verifiers. This would then need to be enforced by all software SURF would provide, the eduID wallet in particular. Thus the wallet would need to know which parties are part of this system and which ones are not in order to be able to enforce this system.

From multiple discussions with other members of the Trust and Identity team, mainly Peter Clijsters and Arnout Terpstra, it seems that the semi-open system seems to be the desired solution by SURF.
If issuers enter a contract for to obtain more privileges, this contract should specify in which claims/credentials they may request and in which situations/contexts.

5.4 Verifiable Data Registry

Verifiable Data Registries are used to mediate the creation and verification of and certain data like identifiers (DIDs), public keys, credential schemas, revocation registries and is also used to maintain this data. The VC specification does not prescribe a specific form such a registry may take, but gives a list of examples, which includes trusted databases, decentralized databases, government ID databases and distributed ledgers and it is even possible to use multiple different verifiable data registries in an ecosystem.
In eduID, the following data would need to be stored on such a registry:

- Public DIDs (e.g. of issuers or individual credentials) as well as their associated public keys and service endpoints. This is the main type of data that will be stored on a VDR.
- Public keys needed to verify signatures (these mostly belong to issuers)
- Credential schemas

- Possibly revocation registries, if utilized

The main challenge here is trying to decide which data structure should be used to implement such a registry. There are essentially two options to consider here:

1. **Blockchain**
   By far the most common data structure employed in existing SSI implementations that functions as a verifiable data registry is a blockchain. Blockchains offer properties such as:

   - Immutability of the data on the chain, meaning that it cannot be changed or altered. Every time someone wants to add a new node/block to the chain, all the other node in the chain first verify its validity and only if the majority of them approves the node gets added to the chain. This also makes it more transparent than other data structures.
   - Each transaction and associated data are visible to all parties within the network, offering transparency.
   - Blockchains are tamper-resistant, as tampering with a block in the blockchain would break the chain and could thus easily be detected. This protects the integrity of the data on it.
   - Blockchains are a decentralized data structure without a central authority maintaining it. Instead a group of nodes maintains the network. This decentralization of course works well with the SSI principles.

   However they are not entirely unproblematic as they are often prone to be heavily restricted in their usage by legislation and, in the case of larger networks, suffer from scalability issues since every node needs to keep a copy of the entire network and each interaction with the blockchain would need to be validated by each node. This can introduce a lot of latency and reduce the throughput. Additionally interactions can be computationally expensive and the infrastructure needed would be very energy-consuming.
   It can make sense to use a public blockchain as SURF could utilize existing infrastructure and at least for the VDR, the parties would not need to trust in SURF, but the blockchain instead. If there was a specific blockchain that is the de-facto standard within SSI, i.e. that would be used by most existing SSI implementations, then you would also get the additional benefit that your systems becomes more interoperable with other SSI implementations (though this alone is not enough to achieve interoperability). The problem with this is of course that every implementation currently uses their own blockchain, which of course makes interoperability a lot harder.

2. **Central Database** Another option would be to use a central trusted database as a verifiable data registry. IRMA for example uses a central server to manage credential schemes. In a similar manner, SURF could host one central eduID database as a verifiable data registry. This would remove the scalability and latency issues of a blockchain and is much easier to manage than a blockchain. However this approach also has its drawback, since it would introduce a central component, which is not ideal with the SSI principle of decentralization, and is of course less than ideal in terms of interoperability. This would also place a lot of responsibility to protect the integrity and availability of all the data on the registry themselves and requires the trust of all other parties involved.

To decide which data structure to use, more research is needed by SURF into blockchains, specifically those that are used in SSI (like e.g. ION (`https://identity.foundation/ion/`), which is used in Microsoft's SSI pilot). The eduID project team would need to discuss this in more detail and examine the possibilities in terms of usable infrastructure.

5.5 Threat model & misuse cases

Aside from all the intended use cases of an eduID SSI, there are certainly some scenarios that one would want to avoid. One of the main threats to an SSI system is a breach of the holders' privacy, which usually means that some party gains access to information (private data ("persoonsgegevens") in particular) that they are not supposed to know/have access to. Thus an abuser/attacker in this context has the main incentive of stealing the data of an eduID. Since some of the information in this SSI ecosystem may contain very privacy-sensitive information, such as bank or credit card data, theft of such data can be highly problematic for eduID users. But even if we are looking at less extreme abuse cases, unintended data access - even to just metadata - can bring about undesirable consequences is terms of privacy such as a user being tracked across different websites.
Example scenarios:

- A verifier, especially of the untrusted kind, asking for certain claims they do not need, trying to exploit users that do not pay much attention to which data is being requested. (e.g. website offering free services to students asking for creditcard/bank data, despite only needing to know if the user is a student or not)
- Verifiers and/or Issuers colluding to track a user across multiple services.
- eduID users being tracked across different services in their interactions, e.g. by someone correlating obtained metadata to one specific user, thereby identifying which websites that user visited.
- Some external party interrupting and or reading messages sent, which they might use later in a replay attack to imitate a holder or just gain access where they are not supposed to using someone else's credentials. In this scenario, not only do we have one victim (the holder), but multiple victims, since verifiers are being tricked here as well. In the absolute worst case, this might also cause reputational damages for issuers, since credentials they issued were misused.

However, the breach of a holder's privacy is not the only threat that should be considered. A similarly important threat is a fraudulent holder or a party maliciously imitating a holder, who is deceiving issuers and/or verifiers. Scenarios where such an attacker/abuser plays a role include:

- Someone tricking a issuer to either issue wrong credentials or the credentials of someone else.
- Attacker forging a credential themselves and tricking verifiers with said forged credentials.
- Holder manipulating a legitimate credential issued to them by changing the data within that credential with the goal of deceiving a verifier.
- Attacker stealing a private identifier and/or private key of an eduID user to imitate them in a specific relationship.

6 Back-end design

After having designed the front-end prototype, the question of how all this needs to be implemented, i.e. what the back-end should look like remains. For SURF, it is very important to make use of open standards as much as possible in the implementation of their products. Therefore I researched suitable open standards for the necessary data models and protocols. Additionally I also provided an overview of all roles and participants within a potential eduID SSI ecosystem as well as the goal architecture. In this section I will provide an overview of the Verifiable Credential Standard and the DID standard, both of which are official W3C standards and are adopted within most existing SSI architectures as the underlying data models. I will explain the basic data model and describe the most interesting features (of Verifiable Credentials). This section will also provide a description of two protocol standards that could be used within eduID in the form of DIDComm and OpenID Connect Self-Issued OP, an extension of the popular OpenID Connect standard that SURF uses within their federated login service SURFconext.

6.1 Verifiable Credentials

6.1.1 Credential Data Model

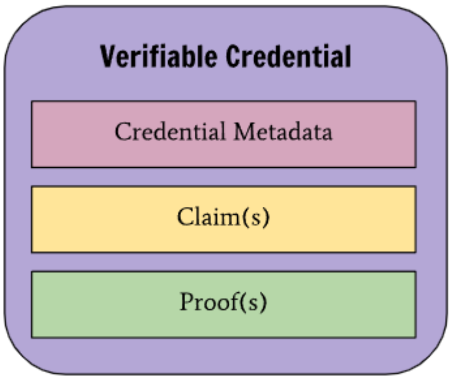


Figure 8: Basic components of a verifiable credential

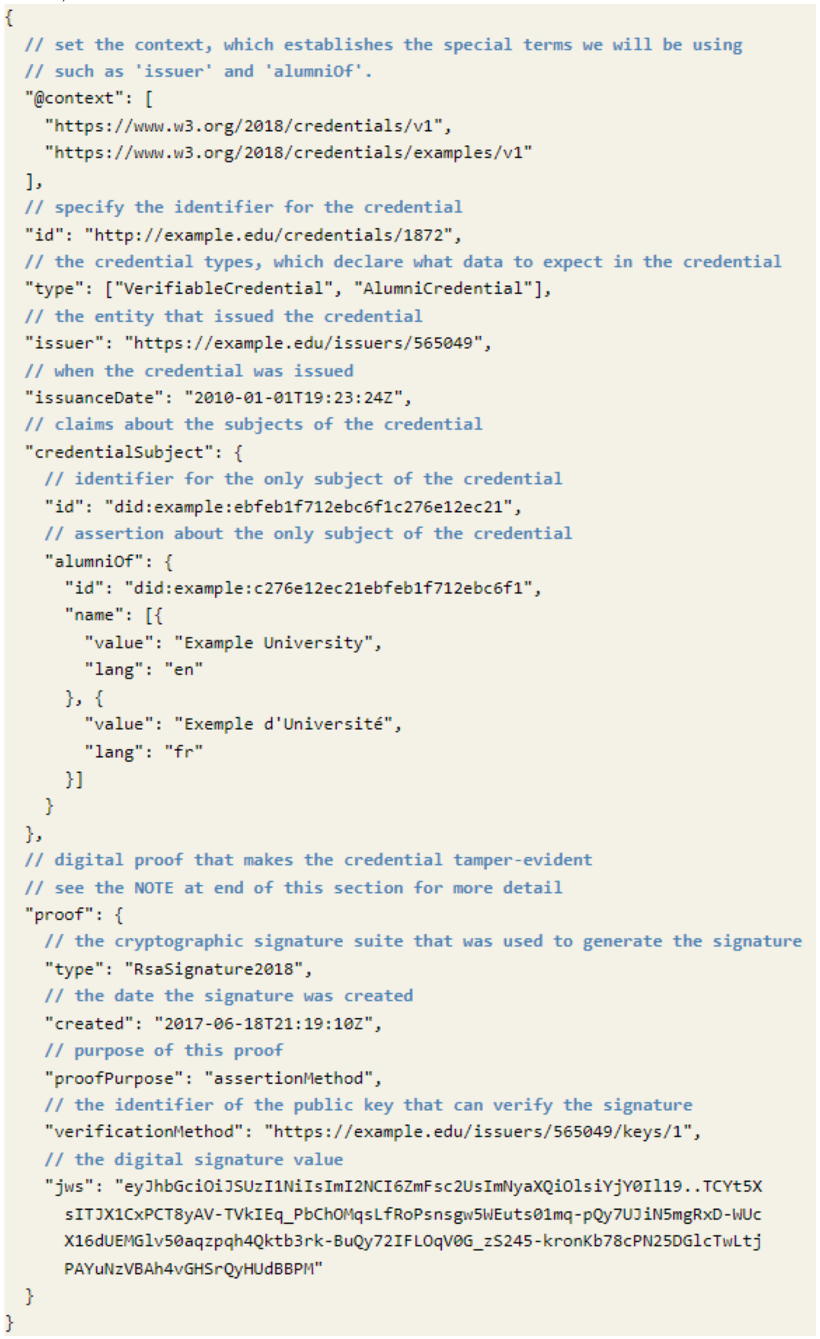
Verifiable Credentials (`https://www.w3.org/TR/vc-data-model/`) is a standard developed within the Verifiable Credentials working group within the W3C and provides a data model for credentials as well as presentations, where users disclose specific claims/attributes, within an SSI system. In comparison to other standards I examined, Verifiable Credentials is one of the more mature standards since it has actually been accepted and recommended by a standards organization, in this case the W3C. Some parts of the specification are still undergoing changes, since very specific requirements are sometimes adapted between different versions. Most of these changes appear to be minor, however it should still be noted that the specification is still receiving regular updates. What the document is severely lacking though is guidelines on the actual implementation of the defined use cases and more advanced concepts, which would be very important to allow for interoperability. So it seems that the standard will still undergo a lot of updates in the future and is thus still at an earlier stage in its life cycle.
As illustrated in figure 8, a Verifiable Credential consists of three basic components:

1. **Metadata:** Metadata "describe[s] properties of the credential, such as the issuer, the expiry date and time, a representative image, a public key to use for verification purposes, the revocation mechanism, and so on." (W3C). They are also cryptographically signed by the issuer of the credential.

2. **Claims:** Claims are statements made about the subject of the credential in the form subject-property-value, e.g. Rob Jansen (subject) - is a student of (property) - Radboud University (value). In IRMA, they are called attributes. Claims are tamper-evident in combination with a proof, i.e. it is easy to detect if a claim has been tampered with.

3. **Proofs:** A proof is data about the identity holder that allows others to verify the source of the data (i.e the issuer), check that the data belongs to the holder (and only them), that the data has not been tampered with, and finally, that the data has not been revoked by the issuer.

A basic example of a Verifiable Credential looks like this (example taken from the specification itself):



```
{
  // set the context, which establishes the special terms we will be using
  // such as 'issuer' and 'alumniOf'.
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://www.w3.org/2018/credentials/examples/v1"
  ],
  // specify the identifier for the credential
  "id": "http://example.edu/credentials/1872",
  // the credential types, which declare what data to expect in the credential
  "type": ["VerifiableCredential", "AlumniCredential"],
  // the entity that issued the credential
  "issuer": "https://example.edu/issuers/565049",
  // when the credential was issued
  "issuanceDate": "2010-01-01T19:23:24Z",
  // claims about the subjects of the credential
  "credentialSubject": {
    // identifier for the only subject of the credential
    "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
    // assertion about the only subject of the credential
    "alumniOf": {
      "id": "did:example:c276e12ec21ebfeb1f712ebc6f1",
      "name": [{
        "value": "Example University",
        "lang": "en"
      }, {
        "value": "Exemple d'Université",
        "lang": "fr"
      }]
    }
  },
  // digital proof that makes the credential tamper-evident
  // see the NOTE at end of this section for more detail
  "proof": {
    // the cryptographic signature suite that was used to generate the signature
    "type": "RsaSignature2018",
    // the date the signature was created
    "created": "2017-06-18T21:19:10Z",
    // purpose of this proof
    "proofPurpose": "assertionMethod",
    // the identifier of the public key that can verify the signature
    "verificationMethod": "https://example.edu/issuers/565049/keys/1",
    // the digital signature value
    "jws": "eyJhbGciOiJSUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19..TCYt5X
      sITJX1CxPCT8yAV-TVkIEq_PbChOMqsLfRoPsnsgw5WEuts01mq-pQy7UJiN5mgRxD-WUc
      X16dUEMGlv50aqzpqh4Qktb3rk-BuQy72IFLOqV0G_zS245-kronKb78cPN25DGlcTwLtj
      PAYuNzVBAh4vGHSrQyHUdBBPM"
  }
}
```

A Verifiable Credential takes the form of a JSON-LD file and features a lot of parameters, most of which are metadata parameters. The ”credentialSubject” parameter contains the actual claims and the ”proof” parameter contains the cryptographic proof needed for the verification of the credential. Both of these parameters are mandatory and always have to be included
The mandatory metadata fields are:

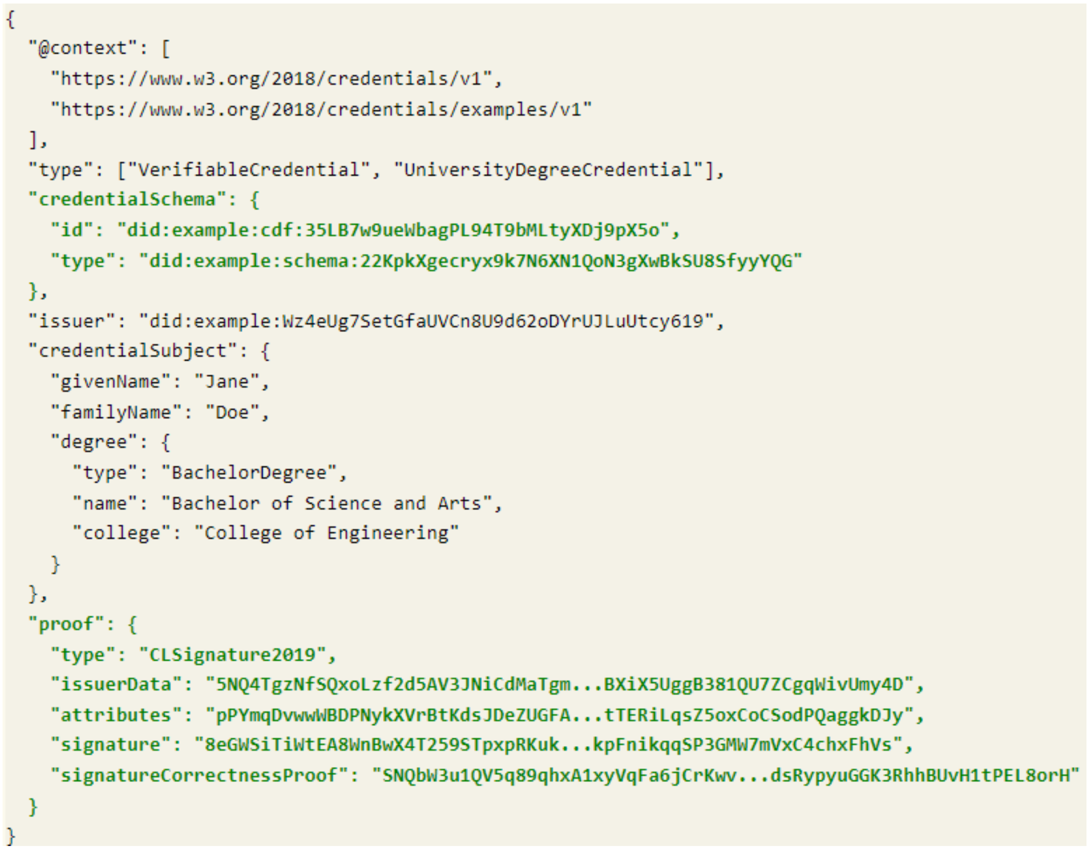
- **@context**
  Contains one or more URIs Used to define the terminology used within the credential. The first URI as seen in the example must always be included and the issuer can add their own contexts if needed.

- **type**
  The type property determines if a VC or verifiable presentation is appropriate in the context it is used. For example a holder might expect to be issued a credential of type "UniversityDegreeCredential", but an issuer sends him a credential of type "SchoolDiplomaCredential". Since this is not what the holder expected, the holder can reject this credential and inform the issuer. Similarly in a verifiable presentation, the type needs to be what the verifier expects to receive, otherwise the verifier will reject it. The value of this property must either be or must map to one or more URIs.

- **issuer**
  Identifies the issuer of the credential.

6.1.2 Verifiable Presentations and Zero-Knowledge Proofs

Similarly to a verifiable credential (VC), there are also verifiable presentations (VP). Similarly to VCs, they also consist of three basic components, namely metadata, verifiable credentials and a cryptographic proof. A presentation can contain an entire credential, but it can also just contain specific claims taken from a credential. A verifiable presentation can even include claims from multiple different VCs of multiple different issuers and can contain claims that are derived from other claims (e.g. a claim that the subject is at least 18 years old might be derived from a claim about the date of birth).
To facilitate this, verifiable presentations often utilize so-called Zero-Knowledge proofs. A Zero-Knowledge Proof is a cryptographic method to prove knowledge of a certain value without disclosing the actual value. A practical example provided by the W3C specification would be "proving that an accredited university has granted a degree to you without revealing your identity or any other personally identifiable information contained on the degree" (W3C).
Zero-Knowledge proofs enable the combination of claims from different verifiable credentials without revealing verifiable credential or subject identifiers, the selective disclosure of single claims instead of having to present an entire credential and the production of derived claims without requiring the involvement of the issuer. This ensures that holders (the users that store the credentials in their digital wallet) only have to disclose the minimum amount of information necessary and nothing more.
To utilize Zero-Knowledge Proofs within a verifiable presentation, issuers need to issue VCs in such a manner that the holder of the credential can derive a proof from the original credential. They should enable holders to "prove the validity of the issuer's signature without revealing the values that were signed, or when only revealing certain selected values" (W3C).
The following is an example of a verifiable presentation using a Zero-Knowledge Proof. The presentation uses a so-called Camenisch-Lysyanskaya Signature, "which allows the presentation of the verifiable credential in a way that supports the privacy of the holder and subject through the use of selective disclosure of the verifiable credential values" (W3C).



```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://www.w3.org/2018/credentials/examples/v1"
  ],
  "type": ["VerifiableCredential", "UniversityDegreeCredential"],
  "credentialSchema": {
    "id": "did:example:cdf:35LB7w9ueWbagPL94T9bMLtyXDj9pX5o",
    "type": "did:example:schema:22KpkXgecryx9k7N6XN1QoN3gXwBkSU8SfyyYQG"
  },
  "issuer": "did:example:Wz4eUg7SetGfaUVCn8U9d62oDYrUJLuUtcy619",
  "credentialSubject": {
    "givenName": "Jane",
    "familyName": "Doe",
    "degree": {
      "type": "BachelorDegree",
      "name": "Bachelor of Science and Arts",
      "college": "College of Engineering"
    }
  },
  "proof": {
    "type": "CLSignature2019",
    "issuerData": "5NQ4TgzNfSQxoLzf2d5AV3JNiCdMaTgm...BXiX5UggB381QU7ZCgqWivUmy4D",
    "attributes": "pPYmqDvwwWBDPNykXVrBtKdsJDeZUGFA...tTERiLqsZ5oxCoCSodPQaggkDJy",
    "signature": "8eGWSiTiWtEA8WnBwX4T259STpxpRKuk...kpFnikqqSP3GMW7mVxC4chxFhVs",
    "signatureCorrectnessProof": "SNQbW3u1QV5q89qhxA1xyVqFa6jCrKwv...dsRypyuGGK3RhhBUvH1tPEL8orH"
  }
}
```

For more information, see `https://www.w3.org/TR/vc-data-model/#zero-knowledge-proofs`.

6.1.3 Interesting concepts/features

Apart from the basic parameters, there are also many more parameters that could be included within a verifiable credential or verifiable presentation. I want to highlight four of them.

- **credentialStatus** The credentialStatus property seems to be the method to implement revocation intended by the W3C. The VC specification defines the property "for the discovery of information about the current status of a verifiable credential, such as whether it is suspended or revoked" (W3C). This property includes a URI to where the status can be checked (e.g. a document on the verifiable data registry). However it does not specify a format for how the check of a credential status should be implemented. At the moment there does not seem to be a standard for this either. Microsoft for example currently uses a proprietary implementation of this status check (see `https://docs.microsoft.com/en-us/azure/active-directory/verifiable-credentials/how-to-issuer-revoke`). I have found two unofficial drafts for status check formats by the W3C Credentials Community Groups in the form of Credential Status List 2017, a very "simple list-based mechanism for publishing and checking the status of a credential" in which every credential and its status is listed in the form of a String (it seems), and Revocation List 2020, which uses a binary list of all credentials an issuer has ever issued in the form of a bitstring with a bit for each credential (0 = valid, 1 = revoked; index of the credential in question provided in credentialStatus property). However both specifications state clearly that they are "not a

W3C Standard nor [...] on the W3C Standards Track".
Most SSI implementations seem to place the burden of checking the revocation status of a credential and making sure the credential is valid on the verifiers. This makes sense of course, since the verifiers can verify the revocation status of the presented claims for themselves and do not have to trust another party to do it for them. Nonetheless, in principle this check could also be performed by the wallet software. However, if this check would only be performed by the wallet, this would change the trust relationship between holder and verifier fundamentally and would require a high degree of trust in the holders (or at least their wallet software) from the verifiers.

- **Data Schemas** Data schemas are used to enforce a very specific structure on a collection of data. They can be used to ensure that credentials of the same type always follow the same template and format. This is needed since, unlike in IRMA for example, the type parameter does not actually give a format for the credential itself, but is more like an unofficial statement about what the credential is supposed to represent. The W3C specification explicitly considers at least two kinds of data schemas:

  1. Data verification schemas
     These can be used to verify that structure and contents of a VC conform to a published schema.
  2. Data encoding schemas
     These are used to map contents of a VC to a different representation format (e.g. a binary format used in a ZKP).

  A schema is added to a VC via an additional credentialSchema property, that specifies its type and an id property (a URI identifying the schema file).

- **Refresh services** Since credentials often expire or sometimes become outdated, it makes sense to include a feature to refresh credentials after expiration or when a new version is available. Verifiable Credentials do have a refreshService property exactly for this purpose, which issuers can use to include a link in the form of an URI to a service where holders can refresh their credentials.
  Unfortunately, there is no standard way on how these refresh services should be implemented yet. However I did find a draft by the W3C Credentials Community Group (see https://digitalbazaar.github.io/vc-refresh-2021/). This draft describes two protocol flows: one for a manual refresh service, where an interactive exchange with the holder is required, and one for an automatic refresh service, that does not require any interactivity with the holder and is fully automatable. The type of the refresh service is defined in the type parameter of the refreshService property.
  Figure 9 shows the general pattern all refresh protocols follow. The figure is taken directly form the specification draft. The protocols defined by this draft are the following:

  - **MediatedRefresh2021 Protocol**
    1. The holder software (wallet) extracts the refreshService.type, refreshService.url, and refreshService.validAfter properties and saves them as type, url, and validAfter, respectively.
       If type does not equal ManualRefresh2021, raise an INVALID_REFRESH_ALGORITHM exception.
       If validAfter is defined, but expresses a datetime that is later than the current datetime, raise a REFRESH_NOT_ALLOWED exception.
       If url is not defined, raise an INVALID_URL exception.

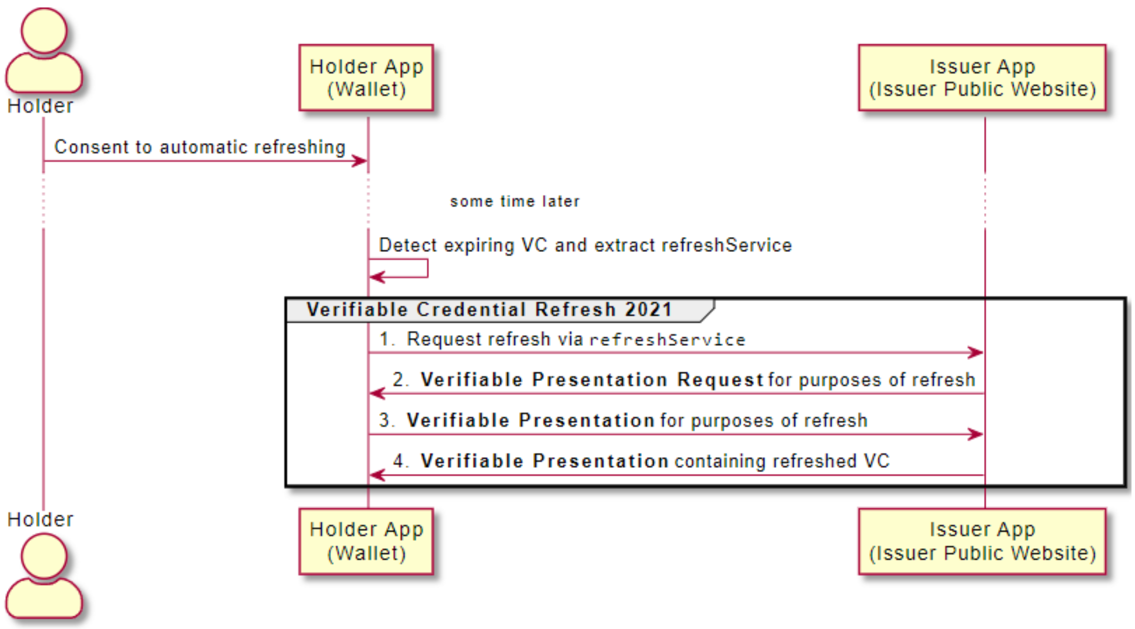

Holder
Holder App (Wallet)
Issuer App (Issuer Public Website)
Consent to automatic refreshing
some time later
Detect expiring VC and extract refreshService
Verifiable Credential Refresh 2021
1. Request refresh via refreshService
2. Verifiable Presentation Request for purposes of refresh
3. Verifiable Presentation for purposes of refresh
4. Verifiable Presentation containing refreshed VC
Holder
Holder App (Wallet)
Issuer App (Issuer Public Website)

Figure 9: Refresh protocol pattern

   2. Open an application, such as a Web browser, that is capable of handling url and pass the value to be opened to the application.

  – **UnmediatedRefresh2021 Protocol**

   1. The wallet extracts the refreshService.type, refreshService.url, refreshService.validAfter, and refreshService.validUntil properties and saves them as type, url, validAfter, and validUntil, respectively.
   2. Use an HTTP client to perform an HTTP GET on url. The response from the server MUST be a Verifiable Presentation Request.
   3. The wallet sends back the requested verifiable presentation. While it isn't made explicit in the draft, I think the wallet needs to present the old credential here to show that is actually has that credential.
   4. The issuer responds with the re-issued VC.

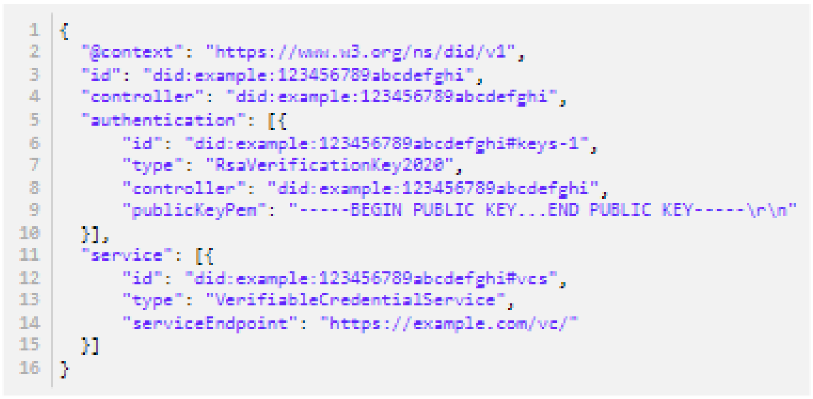
- **Terms Of Use** The W3C specification defines a termsOfUse property that can be included by issuers or holders to communicate actions holders and/or verifiers must (obligation), may (permission) or must not (prohibition) perform when accepting the credential/presentation. The specification provides an example, where this property could be used by government-issued verifiable credentials to instruct digital wallets to limit their use to similar government organizations. It could thus potentially be used to limit which verifier is allowed to process the credential and the claims within it. The problem is of course that verifiers could just choose to ignore the property altogether and process the claims anyway. The only way this can actually be enforced is if it is enforced within the implementation of the wallet itself, since that is the component that can actually be controlled here.

6.2 Decentralized Identifiers

Decentralized Identifiers, better known as DIDs, are type of globally unique identifiers that are generated by entities themselves, of which they prove control via cryptographic proofs such as digital signatures. According to the DID specification, the difference between DIDs and regular, federated identifiers is that ”DIDs have been designed so that they may be decoupled from centralized registries, identity providers, and certificate authorities” (W3C), thus users do not need the permission of an external party to prove ownership of their identifier. Each entity can have as many DIDs as they desire to separate their identity between different context. As the name also suggests, these identifiers are also decentralized, which means that there is no dependence on a central authority as all parties create and maintain their identifiers themselves. The illustrations on figure 10, taken directly from the W3C specification, show the architecture of a DID and provides an example of what a DID looks like. However unlike the chart suggests, DIDs are not necessarily bound to a ledger (especially not those that are not supposed to be public), but can really be recorded on any desired data structure. As the illustration shows a DID consists of three parts:

1. The did URI scheme identifier
2. The DID method used to resolve the DID
3. The DID method-specific identifier

Using a so-called DID resolver, each DID can be resolved with its corresponding DID method to a DID document, that contains information associated with that DID, e.g. such as cryptographic public keys necessary to verify the DID and services relevant to the interaction with DID subjects. The following illustration[1] provides an example of what a DID document could look like:



```
{
  "@context": "https://www.w3.org/ns/did/v1",
  "id": "did:example:123456789abcdefghi",
  "controller": "did:example:123456789abcdefghi",
  "authentication": [{
      "id": "did:example:123456789abcdefghi#keys-1",
      "type": "RsaVerificationKey2020",
      "controller": "did:example:123456789abcdefghi",
      "publicKeyPem": "-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n"
  }],
  "service": [{
      "id": "did:example:123456789abcdefghi#vcs",
      "type": "VerifiableCredentialService",
      "serviceEndpoint": "https://example.com/vc/"
  }]
}
```

Figure 8: Example of a DID document (JSON-LD)

These DID documents are controlled by their so called DID controllers. The controller is the only entity allowed to make changes (as defined by the DID method) to the DID document, however a DID might have multiple controllers and the DID subject can also act as a DID controller.
DIDs can also be private (i.e. only used between two parties (also only known by those two parties) to create a secure private channel for themselves) or public (i.e. they are publicly available). Public DIDs are often used to start exchanging private DIDs. Private DIDs could be used in eduID as identifiers of eduID users within relationships with services, such that they have a

[1]Source: `https://www.fit.fraunhofer.de/content/dam/fit/de/documents/Fraunhofer%20FIT_SSI_Whitepaper_EN.pdf` p.20

different identifier for each service.

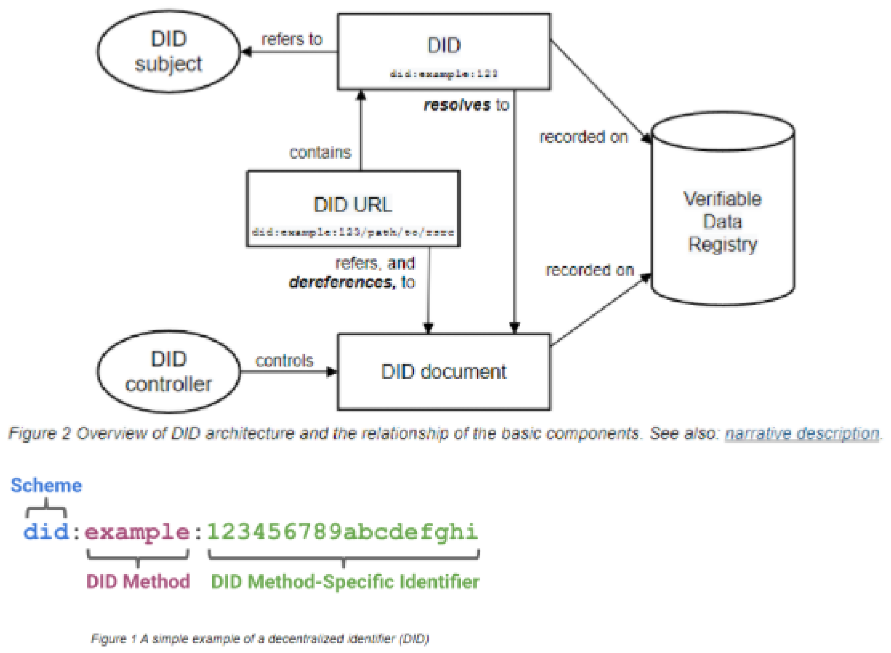

DID subject
refers to
DID
did:example:123
resolves to
recorded on
contains
DID URL
did:example:123/path/to/rsrc
Verifiable Data Registry
refers, and dereferences, to
recorded on
DID controller
controls
DID document
Figure 2 Overview of DID architecture and the relationship of the basic components. See also: narrative description.
Scheme
did:example:123456789abcdefghi
DID Method
DID Method-Specific Identifier
Figure 1 A simple example of a decentralized identifier (DID)

Figure 10: DID architecture (top) and example DID (bottom)

6.3 Protocol standards

After having examined the core data models (VCs and DIDs) and the SSI ecosystem, one aspect that still remains to be covered in the communication between the participants and the implementation of use case like revocation and credential refreshing. Unfortunately, there are no widely adopted and mature standards yet for communication protocols and some of the more advanced concepts of Verifiable Credentials. However, some standards do exist here, however they are still lacking in their adoption and partly in their maturity. For the purpose of this internship I examined two standards in particular, that seem to be very promising for the future.

6.3.1 DIDComm

The first standard I examined was DIDComm. DIDComm originated in the open-source Aries project, which is hosted on Hyperledger Indy, and has since been adopted by a working group within the Digital Identity Foundation (DIF). It is designed as messaging protocol for DID-based relationships. For the purpose of eduID, I did not delve into the "proper" DIDComm messaging specification itself (see `https://identity.foundation/didcomm-messaging/spec/`), but rather specific protocols that were built on top of the specification and are defined as RFCs within the Aries project.
At its core, DIDComm creates a secure private communication channel between any two participants by exchanging pairwise identifiers in the form of DIDs over which they can then exchange further messages. While DIDComm aims to cover a large number of use cases, but the core protocols mainly cover the use cases connecting two parties, requesting and issuing credentials

and proving certain properties with credentials. The RFCs give an idea of the messages that can and need to be exchanged and provides a general schema/structure for the messages, however a lot of the message construction is still left up to the implementer.
DIDComm protocols have been adopted by Sovrin, IOTA and some smaller blockchain-based SSI implementations (however it is hard to find exact information on who actually adopted DIDComm). However DIDComm is not a mature standard yet and is partly still in the RFC and draft stages. It is thus not yet an officially adopted standard by the DIF yet.
The protocols are the following:

- **Relationship Establishment** (`https://github.com/hyperledger/aries-rfcs/blob/main/features/0023-did-exchange/README.md`)

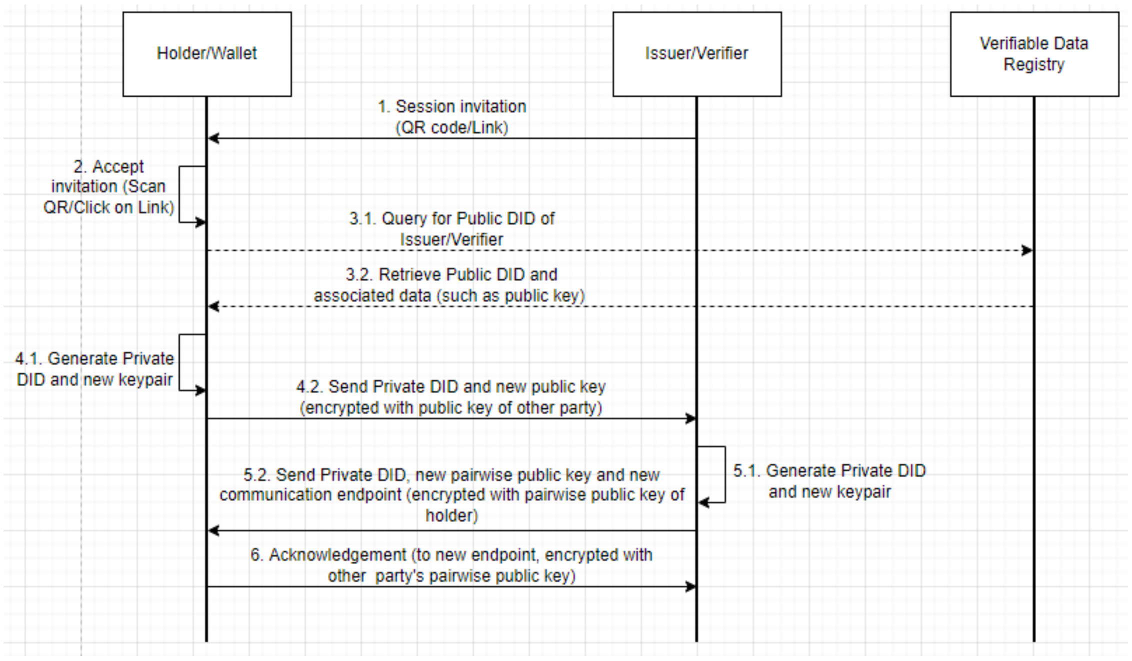

Holder/Wallet
Issuer/Verifier
Verifiable Data Registry
1. Session invitation (QR code/Link)
2. Accept invitation (Scan QR/Click on Link)
3.1. Query for Public DID of Issuer/Verifier
3.2. Retrieve Public DID and associated data (such as public key)
4.1. Generate Private DID and new keypair
4.2. Send Private DID and new public key (encrypted with public key of other party)
5.1. Generate Private DID and new keypair
5.2. Send Private DID, new pairwise public key and new communication endpoint (encrypted with pairwise public key of holder)
6. Acknowledgement (to new endpoint, encrypted with other party's pairwise public key)

Figure 11: Overview of the DIDComm Relationship Establishment protocol

Regardless of if the holder has interacted with the other party before or not, a connection/session needs to be established between the two first. The first step here is for the issuer/verifier the holder wishes to interact with to offer some sort of "invitation" to start a session. This invitation should take the form of a QR code or a some sort of link (e.g. similar to how you can add cards in the IRMA app by clicking on a link within the app). The user then needs to accept this invitation, either by scanning the QR code with the eduID wallet app or by clicking on the link.
In the Sovrin implementation, if the holder is interacting with the issuer/verifier for the first time, the wallet will then query the verifiable data registry for the public DID of the issuer/verifier such that the wallet can obtain necessary information such as the issuer's/verifier's public (verification) key, their endpoint and authentication information.
Instead of an explicit invitation, a sort of implicit invitation may also be used. As opposed to an explicit invitation, where the issuer/verifier sends an explicit invitation message to the wallet, an implicit invitation in this context means that the wallet already knows the

Public DID and associated data of the other party (e.g. because they published that data), thereby skipping the prior steps and immediately sending an exchange request (Step 4 in the diagram).
If "untrusted" verifiers do not put their public DIDs on the verifiable data registry, then this information needs to be communicated differently to the user. One approach here would be to have those verifiers send their public DID directly to the wallet. This approach does have an inherent risk of a Man-in-the-Middle attack, however one might argue that the damage this would do is not very high, as "untrusted" verifiers can only request non-privacy-sensitive claims like "is student" or "is at least 18 years old" anyway. One risk this does have is that the wallet would establish a permanent private channel with the MitM instead of the intended verifier such that when they scan a QR code or click on a link of that verifier, they will always use that "fake" channel, so that holder effectively cannot interact with the intended verifier as long as that channel exists.
The wallet then either generates a private DID and keypair exclusive for this relationship or lets a central eduID component generate these for them (depending on how private DIDs should be managed within the ecosystem). The wallet then sends an encrypted (with the issuer's/verifier's public key) message containing this private DID and its associated DID document that contains the encryption ("public") key of the newly generated keypair to the issuer/verifier.
The issuer/verifier then also generates a private DID and keypair for this relationship and sends a message - encrypted with the holder's encryption key - containing their private DID, its associated DID document, that contains the encryption key of the newly generated keypair to the holder, and an endpoint that is to be used for all subsequent interactions between the two parties. It might then be a good idea for the holder to send an acknowledgment message to the issuer/verifier over their private channel (to the specified endpoint and encrypted with the other party's pairwise public key) to properly complete this part of the exchange.
If these parties have interacted before, then they do not need to establish this private channel as it already exists. In this case, after scanning the QR code or clicking on the link, the wallet of the holder should now recognize which party they are going to be interacting with and thus now knows which channel (aka which private DID and keypair) it should use for this session and vice versa. The actual session is then initiated by the issuer/verifier.

- **Credential Issuance** (`https://github.com/hyperledger/aries-rfcs/blob/main/features/0036-issue-credential/README.md`)
  Figure 12 provides an overview of the protocol, taken directly from its RFC. The messages in this protocol are sent over the private channel established in the relationship establishment protocol.
  Optionally, the holder might initiate the protocol with a message that specifies what kind of credential they would like to receive. They may also send this message as a response to a Credential Offer to request adjustments to the offered credential data. However, usually Issuers send a credential offer message to the holder to initiate the protocol. That message describes what kind of credential they intend to offer and, if they expect to be paid a fee, the message also specifies the amount of that fee. In Hyperledger Indy, this message is required, because it forces the Issuer to make a cryptographic commitment to the set of fields in the final credential and thus prevents Issuers from inserting spurious data. In Sovrin this references a specific credential schema/claim definition on the Sovrin ledger (Sovrin's Verifiable Data Registry), which users may optionally verify and retrieve more information about by querying the verifyable data registry for this credential schema.

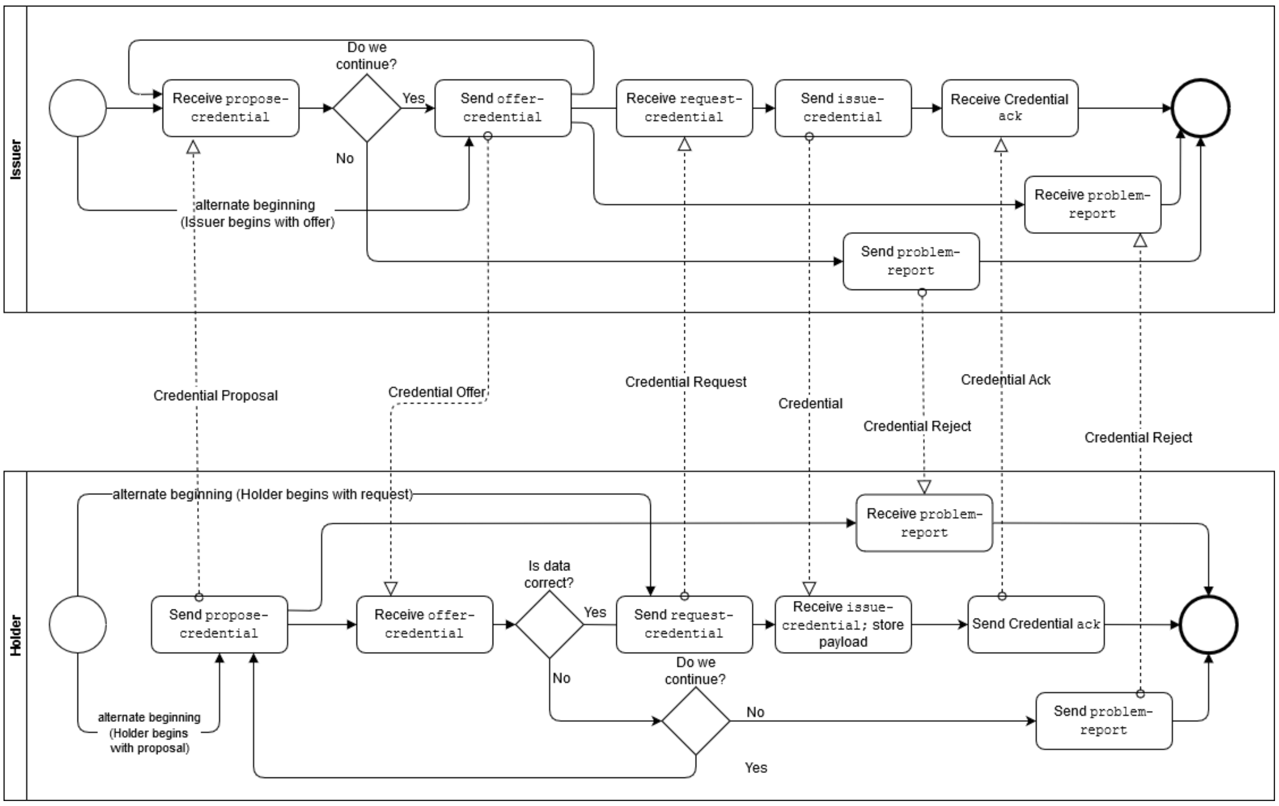


Figure 12: Overview of the DIDComm Credential Issuance protocol

After that, the Holder sends a Credential Request to the Issuer to request the issuance of a credential. They might also specify which specific credential formats they request. In Sovrin, this request also references the prior credential offer and contains a blinded link secret and blinded proving secret.
In response to this, the issuer sends the holder a response message with an attached payload containing the credentials beings issued. In Sovrin, the issued credentials contain the blinded secrets from the User, a number of attributes and a reference to the CredentialDefinition on the verifiable data registry and is signed with the Issuer's Public DID. If the issuer wants an acknowledgement that the issued credential was accepted, this message must be decorated with an additional ~please-ack parameter, and it is then best practice for the new Holder to respond with an explicit ack message (see also Aries RFC 0015).

- **Presenting Proof** (`https://github.com/hyperledger/aries-rfcs/blob/main/features/0037-present-proof/README.md`)
  Figure 13 provides an overview of the protocol, taken directly from its RFC. The messages in this protocol are sent over the private channel established in the relationship establishment protocol.
  Usually, the process of credential presentation is initiated by the Verifier requesting the Holder to cryptographically proof some attributes about themselves. This is done via the presentation request message which describes values that need to be revealed and predicates that need to be fulfilled. The verifier may also request those attributes to come from

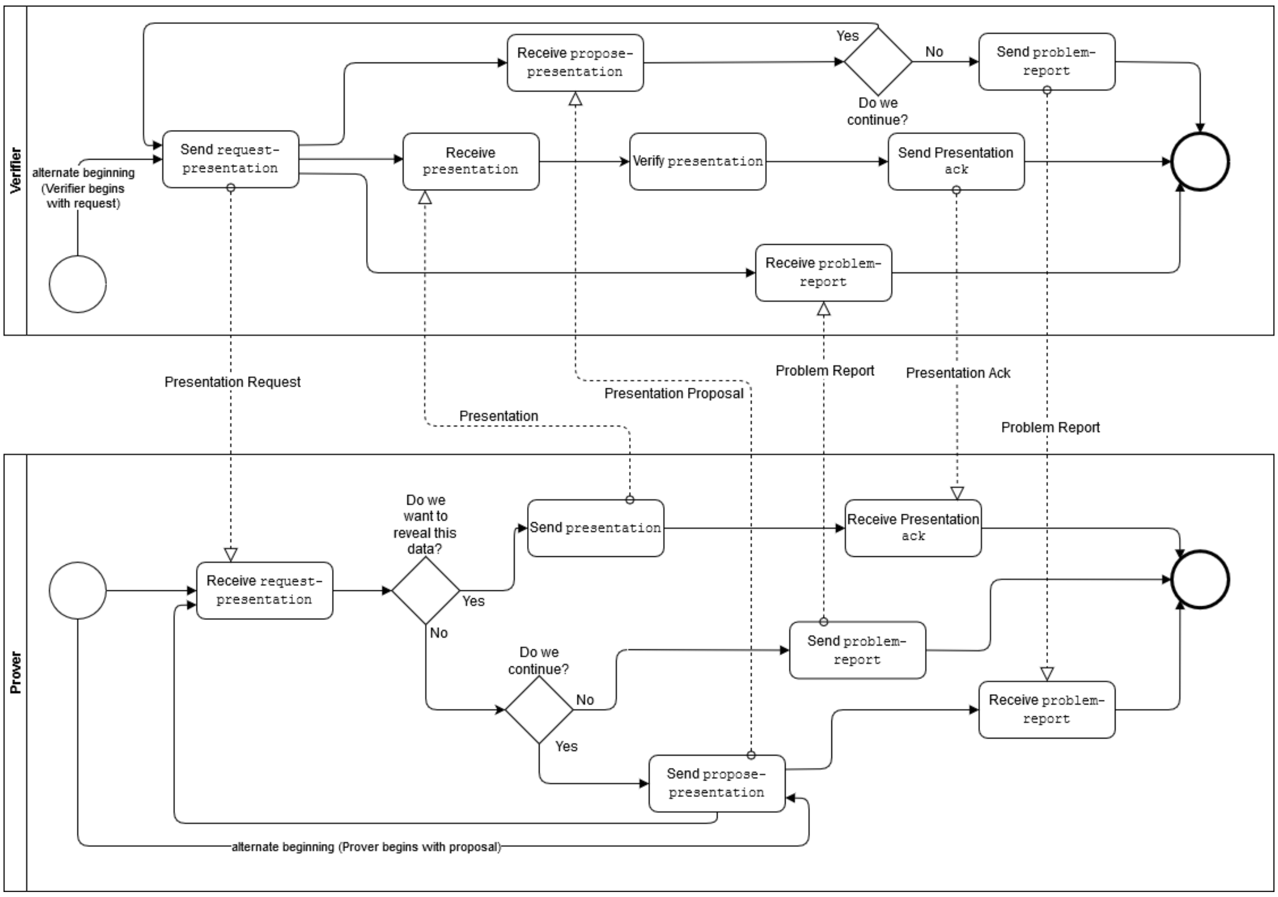


Figure 13: Overview of the DIDComm Presenting Proof protocol

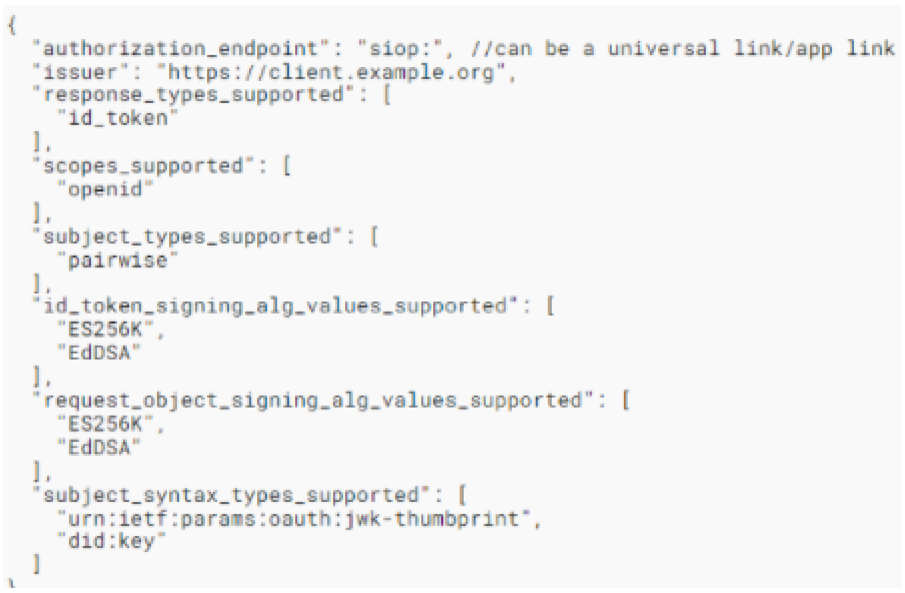
specific credentials (/schemas) and/or specific issuers.
If the holder wishes to disclose this data to the verifier, they respond by sending a verifiable presentation containing the requested claims and all necessary proofs in a presentation message. Upon reception of this message, the verifier verifies this data and may send an ack message back to the holder upon successful verification. In Sovrin, the verifier also checks if the claims have been revoked while verifying the presentation. This is done by checking the verification proof against the corresponding accumulator on the Sovrin ledger (the VDR). If these two values do not match, then the claim (or rather its underlying credential) is no longer valid.

6.3.2 OpenID Connect Self-Issued OP

The other standard I examined was OpenID Connect, or rather extensions to OpenID Connect to add a self-issued OP, such that users do not have to rely on an external party as their identity provider, and support for verifiable presentations to replace the claims parameter within OIDC as well as a draft to issue verifiable presentation. Microsoft is playing a big part in the development and specification of these extensions and are at the moment the main party using OIDC within Self-Sovereign Identity.

The base extension (Self-Issued OpenID Provider v2; see `https://openid.net/specs/openid-connect-self-issue` `0.html`) defines mechanisms on how an RP (issuer or verifier) can invoke and discover a self-issued OP, defines a new ID token for self-issued OPs and defines a basic request-response protocol. This protocol is extended by the OpenID Connect for Verifiable Presentations specification (see `https://openid.net/specs/openid-connect-4-verifiable-presentations-1_0.html`), which adds a so-called vp_token to the self-issued OP ID token. This new token contains either a request for a verifiable presentation or a verifiable presentation.
Credential Issuance is not covered by the base specification, but there is a draft for a credential issuance protocol with OIDC in the OpenID Connect repository on Bitbucket (see `https://bitbucket.org/openid/connect/src/master/individual/draft-lodderstedt-openid-connect-4-credential` `0.md`). This draft is unfortunately still incomplete and is still missing entire sections, however the protocol and the messages are already clearly defined. As opposed to the DIDComm protocols, the OIDC specifications very clearly define the messages sent in great detail and include multiple examples to further illustrate the messages.
In terms of maturity, the base SIOP specification and its extension for Verifiable Presentations are on the Standards Track within the OpenID Foundation. They also use a lot of the already existing infrastructure of OpenID Connect.
For the sake of this section, I will not describe processes that are part of the basic OpenID Connect standard, but only those that are added with the SIOP extension.

- **Self-Issued Open ID Provider Invocation & Discovery + Relying Party Registration (Relationship establishment)**
  Before starting the "proper interaction", in OIDC the parties involved need to exchange



```
{
  "authorization_endpoint": "siop:", //can be a universal link/app link
  "issuer": "https://client.example.org",
  "response_types_supported": [
    "id_token"
  ],
  "scopes_supported": [
    "openid"
  ],
  "subject_types_supported": [
    "pairwise"
  ],
  "id_token_signing_alg_values_supported": [
    "ES256K",
    "EdDSA"
  ],
  "request_object_signing_alg_values_supported": [
    "ES256K",
    "EdDSA"
  ],
  "subject_syntax_types_supported": [
    "urn:ietf:params:oauth:jwk-thumbprint",
    "did:key"
  ]
}
```

Figure 14: Example set of dynamic OP metadata

  metadata, which is essentially the equivalent to a relationship establishment as described for DIDComm above. In contrast to regular OIDC, the Relying Party (in this case a Issuer or Verifier, though from the specification it seems this is mainly Verifiers) does not have an API endpoint to reach the user since they use a Self-Issued OP instead of a regular one, which also means that both parties have to exchange metadata at every single request.

This process consists of two steps namely the invocation and discovery of the Self-Issued OP (SIOP, the wallet) and the registration of the relying party (RP, issuer or verifier) by the holder, or rather their wallet.

Self-Issued Open ID Provider Invocation & Discovery
The RP has no reliable way of figuring out how/where to reach the Self-Issued OP of the Holder they are interacting with, so they need to determine where to direct their request first. When sending the request, the RP essentially has two ways to reach and invoke the actual application (aka the wallet) that can process their request:

– Encoding the request in a QR code or deep link, which the end-user then either needs to scan with their wallet app (QR code) or click/tap on (deep link).

– Including an authorization_endpoint of a Self-Issued OP within the request, which will automatically open a specific target application.

Option 1 would be the most sensible for eduID and seems to be the standard way these things are handled within SSI. For security purposes, the message also contains a nonce to prevent replay attacks.
The RP obtain the authorization_endpoint of the wallet, to which it can send targeted requests either through a static or a dynamic set of Self-Issued OP metadata. Static configurations are used when the RP has no means to pre-obtain Self-Issued OP Discovery Metadata, hence it is less relevant to eduID. The RP can obtain a dynamic set of Self-Issued OP metadata through the regular OpenID Connect Discovery Protocol (see this specification: `https://openid.net/specs/openid-connect-discovery-1_0.html`) or some out-of-band mechanism at this stage. The specification includes an example set of dynamic OP metadata, which can also be seen in figure 14. To utilize pairwise/private DIDs as identifiers, the parameter "subject_types_supported" needs to be "pairwise" and the parameter "subject_syntax_types_supported" needs to include "did:[supported did method]" for all supported DID methods. "authorization_endpoint" tells the RP where the wallet can be reached (usually via an App Lnk or a Universal Link) and ""issuer" "MUST be identical to the iss Claim value in ID Tokens issued from this Self-Issued OP" (OpenID Connect Workgroup).

Relying Party Registration
Relying part registration is usually done as part of the first step of the "proper" SIOP protocol, the Self-Issued OpenID Provider Authentication Request. How this is done depends on if the RP is pre-registered with the OP (the wallet) or not.
If the Holder has pre-registered the RP in question (e.g. via the regular OIDC Registration Protocol), the client_id needs to be equal to the client identifier the RP has obtained from the Self-Issued OP during pre-registration, and registration nor registration_uri parameters must not be present in the Self-Issued OP Request. Furthermore, the public verification key must be obtained during the pre-registration process in the case that the Self-Issued OP Request is signed.
Example of a same-device request with pre-registered RP (from the specification):

```
HTTP/1.1 302 Found
Location:  https://client.example.org/universal-link?
  response_type=id_token
  &client_id=s6BhdRkqt3
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
  &scope=openid%20profile
  &nonce=n-0S6_WzA2Mj
```

A non-pre-registered RP needs to dynamically register with the Self-Issued OP (/the wallet app). The mechanism for this depends on if the Self-Issued OpenID Provider request was signed or not. Assuming it was signed, then the public key needed to verify the signature can be obtained from the RP's client ID. Depending on the chosen Relying Party Metadata Resolution Method, the rest of the RP Registration metadata SHOULD be included either in the registration parameter inside the Self-Issued OP request (either directly or by reference via registration_uri (not both)), or in the Entity Statement as defined in OpenID Federation 1.0 Automatic Registration.

There are essentially two Relying Party Metadata Resolution Method. The first one (OpenID Federation 1.0 Automatic Registration) is less interesting for eduID than the second one, Decentralized Identifier Resolution, where the Relying party's client_id is a did. Then the wallet has to obtain the verification key from the associated DID document (this needs to be identified by the kid in the header of the DID document). Other RP metadata has to be obtained from the registration parameter (as also stated earlier).

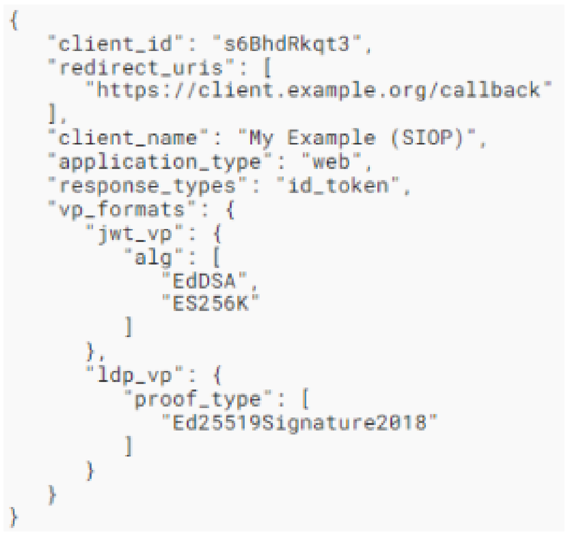
Example of a client_id resolvable using Decentralized Identifier Resolution:

```
"client_id": "did:example:EiDrihTRe0GMdc3K16kgJB3Xbl9Hb8oqVHjzm6ufHcYDGA"
```

Example of a signed cross-device request when the RP is not pre-registered with the Self-Issued OP and uses Decentralized Identifier Resolution:

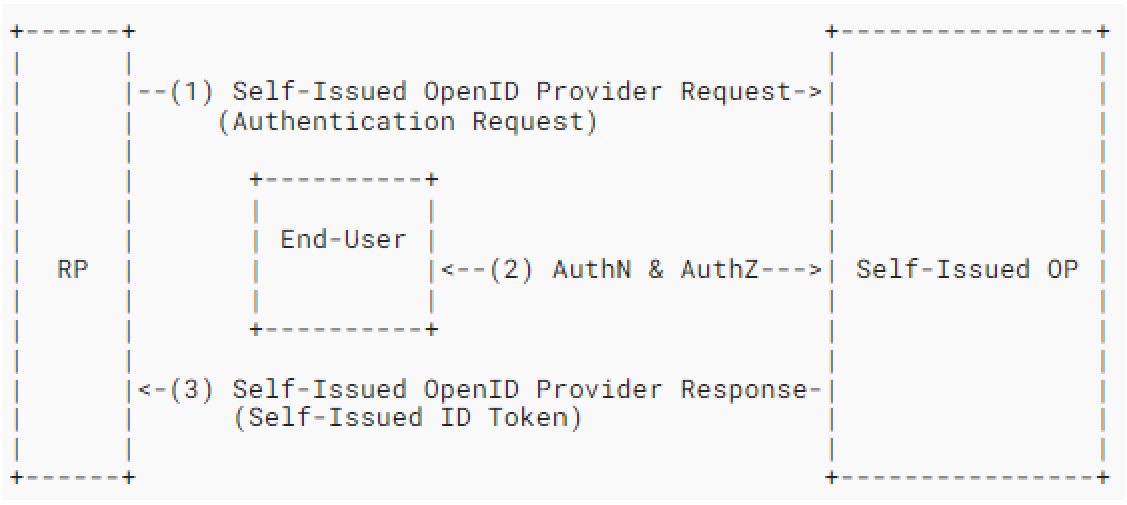
```
openid://?
  scope=openid%20profile
  &response_type=id_token
  &client_id=did%3Aexample%3AEiDrihTRe0GMdc3K16kgJB3Xbl9Hb8oqVHjzm6ufHcYDGA
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
  &claims=...
  &registration=%7B%22subject_syntax_types_supported%22%3A
  %5B%22did%3Aexample%22%5D%2C%0A%20%20%20%20
  %22id_token_signing_alg_values_supported%22%3A%5B%22ES256%22%5D%7D
  &nonce=n-0S6_WzA2Mj
```

To support the use of verifiable presentations, a new parameter "vp_formats" is added to the RP's metadata to define the formats, proof types and algorithms of verifiable presentation and verifiable credential that this RP supports (valid values (others may be added): jwt_vp, ldp_vp, jwt_vc and ldp_vc). Example for an RP registering with a SIOP with the registration request parameter:



```
{
   "client_id": "s6BhdRkqt3",
   "redirect_uris": [
      "https://client.example.org/callback"
   ],
   "client_name": "My Example (SIOP)",
   "application_type": "web",
   "response_types": "id_token",
   "vp_formats": {
      "jwt_vp": {
         "alg": [
            "EdDSA",
            "ES256K"
         ]
      },
      "ldp_vp": {
         "proof_type": [
            "Ed25519Signature2018"
         ]
      }
   }
}
```

- **Disclosing claims (the "proper" OIDC SIOP protocol)**
  Figure 15 provides an overview of the SIOP protocol, which is essentially a simple request-



```
+------+                                           +----------------+
|      |                                           |                |
|      |--(1) Self-Issued OpenID Provider Request->|                |
|      |     (Authentication Request)              |                |
|      |                                           |                |
|      |       +----------+                        |                |
|      |       |          |                        |                |
|      |       | End-User |                        |                |
|  RP  |       |          |<--(2) AuthN & AuthZ--->| Self-Issued OP |
|      |       |          |                        |                |
|      |       +----------+                        |                |
|      |                                           |                |
|      |<-(3) Self-Issued OpenID Provider Response-|                |
|      |      (Self-Issued ID Token)               |                |
|      |                                           |                |
+------+                                           +----------------+
```

Figure 15: Overview of the OIDC SIOP protocol

response protocol. In a scenario where multiple devices are used, the Self-Issued OpenID Provider Request needs to be rendered as a QR code that the user can scan with their wallet app on their phone.

Self-Issued OpenID Provider Authentication Request
Figure 16 gives an example of an SIOP Authentication Request in a same-device flow. This request must always contain a nonce for basic replay protection. The "registration" parameter is used for the registration of the RP, if needed.
For eduID, the most important parameter here is the "claims" parameter. RPs use the claims parameter to request specific claims from Holders. When requesting claims in the form of a verifiable credential or a verifiable presentation, the userinfo subparameter is

```
HTTP/1.1 302 Found
Location: openid://?
  scope=openid
  &response_type=id_token
  &client_id=https%3A%2F%2Fclient.example.org%2Fcb
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
  &claims=...
  &registration=%7B%22subject_syntax_types_supported%22%3A
  %5B%22urn%3Aietf%3Aparams%3Aoauth%3Ajwk-thumbprint%22%5D%2C%0A%20%20%20%20
  %22id_token_signing_alg_values_supported%22%3A%5B%22ES256%22%5D%7D
  &nonce=n-0S6_WzA2Mj
```

Figure 16: Example SIOP Authentication Request

replaced with a so-called vp_token. This vp_token contains a presentation_definition parameter (which is defined in great detail in the DIF Presentation Exchange specification) and specifies what kind of claims or credential types are being requested.
The important property within the presentation_definition are the "input_descriptors". These specify the requested claims and what types of patterns (i.e. credential schemas) they may come from.
Figures 18 and 19 (see Appendix) provide two examples of what this claims parameter could look like in practice.

Self-Issued OpenID Provider Authentication Response This response is an OpenID Connect Authentication Response generally made in the same manner as in the Implicit Flow, as defined in Section 3.1.2.5 of of the core OpenID specification, with some exceptions. For example a vp_token, that contains either a single verifiable presentation or an array of verifiable presentations, MUST be provided in the same response as the id_token of the respective OpenID Connect transaction.
Each of those verifiable presentations may contain a presentation_submission element, that links the input_descriptor elements from the authentication request to the respective verifiable presentations within the vp_token along with format information. The root of the path expressions in the descriptor_map is the respective verifiable presentation, pointing to the respective Verifiable Credentials (for an example see figure 20 in the appendix).
The OP may also add a _vp_token element within the id_token, that only contains a presentation_submission element that links the input_despriptor identifiers from the authentication request to the respective verifiable presentations within the actual vp_token along with additional information about the format. This element is mostly used when the holder/wallet (OP) wants to provide the RP with additional information about the format and structure before the RP processes the vp_token. For examples of what this looks like in practice see figures 21 and 22 in the appendix.
In a same device flow (user accesses RP on phone), the response parameters will be returned in the URL fragment component, unless a different Response Mode was specified. In a cross-device flow, the Self-Issued OP directly sends a HTTP POST request with the authentication response to an endpoint exposed by the RP.

Possible Extension - Transparency about storage
In some cases, verifiers may want to store some of the claims they received. For example, when a student presents a edubadge claim, that proves they passed a certain course at a different university, to their home university, the university needs to store that data. In these cases it would be nice to get some transparency about this storage of specific

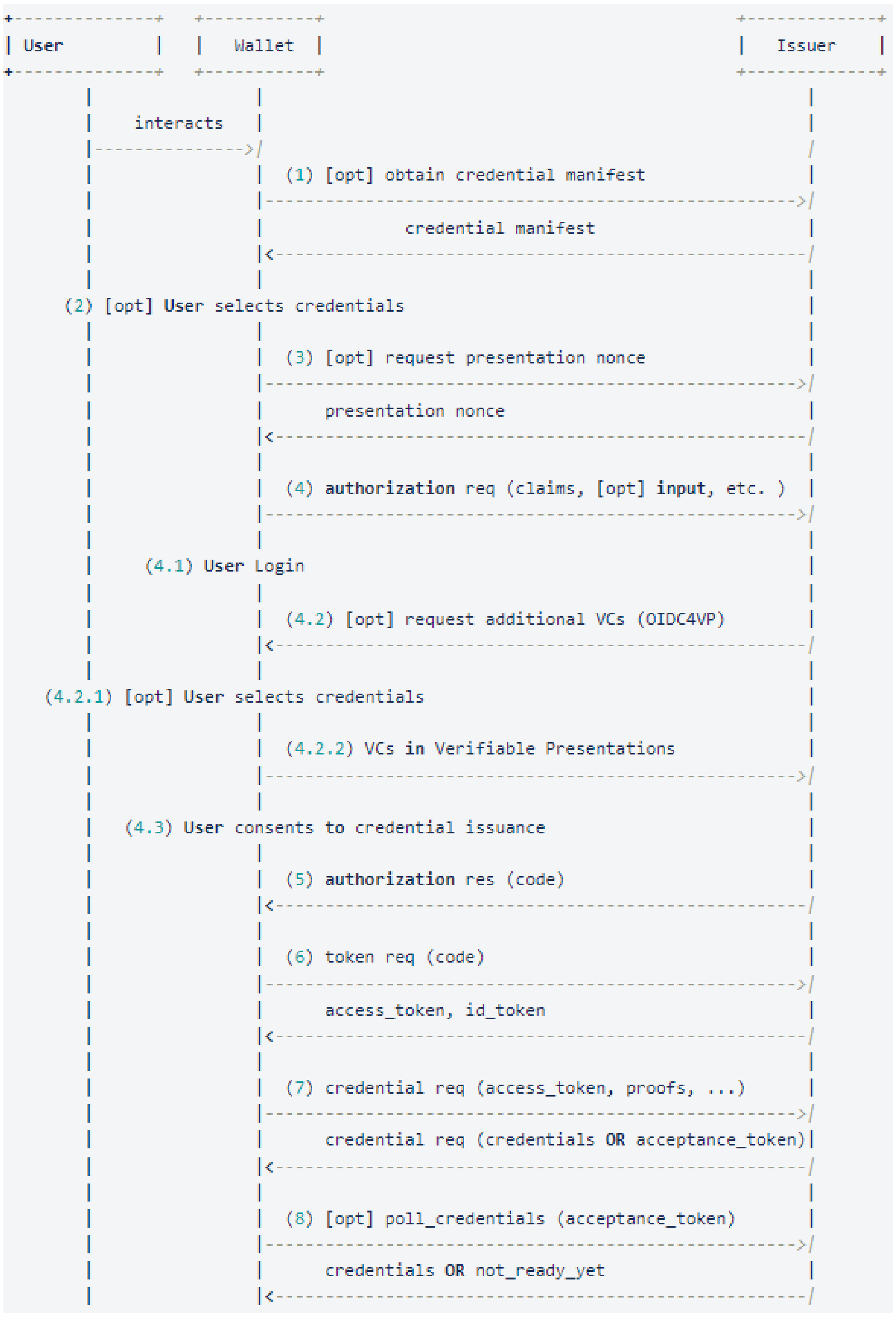
claims and about how long these claims are going to be stored (obviously, in the case of an edubadge this will be rather long term since it is relevant for the eventual degree). The basic versions of these protocols do not seem enable this transparency, so they would need to be extended for this purpose.
I see essentially two ways the protocols can be expanded for this purpose. The first way would be to include an additional parameter in the verifier's request to the holder. This parameter should then state if claims are being stored and if yes, for how long, i.e. it should state a timestamp when the storage period will end (similarly to the expiry time of credentials). The advantage of this approach would be that the protocols remain largely unchanged since only one parameter is added to one message, preserving the general structure of the protocols. However the wallet would need to keep this data in mind and it cannot be verified afterwards.
The second way would be add an additional signed message at the end from the verifier to the holder (or in the case of DIDComm add it to the ack message) that contains a receipt for the holder, that lists what data has been received, when this data has been received, what has been stored and until when the stored claims will be stored. This would of course require modifying the protocol flow a bit, however with a receipt the data within the receipt can now be verified and the wallet can keep better track of past interactions.
One thing to note here that is impossible to force verifiers (especially "untrusted" ones) to be this transparent about data storage, since they are not legally required to.

Possible Extension - Transparency about why claims are being requested
Another possible extension would be to include an additional string parameter in the verifier's request message to the holder to give insight into why the requested attributes are actually being requested. Again, it is not possible to force untrusted verifiers to include a sufficient explanation, as again they are not required to by law. However for trusted verifiers that do enter into an eduID trust framework, this could be required in the contract they sign.

- **Credential Issuance** (`https://bitbucket.org/openid/connect/src/master/individual/draft-lodderstedt-openid-connect-4-credential-issuance-1_0.md`)
  Figure 17 provides an overview of the protocol defined by the draft. The starting point of the protocol is a user interaction with the wallet app, where the user either wants to present a credential and has no suitable credential present in their wallet or has visited the web site of a Credential Issuer and wants to obtain a credential from that issuer.
  The steps are defined as follows (more or less directly copied from the specification draft):

  1. (OPTIONAL) Before the proper issuance, the wallet might first obtain a credential manifest from the issuer (see https://identity.foundation/credential-manifest/), that defines which Verifiable Credentials the Issuer can issue, and optionally what kind of input from the user the Issuer requires to issue that credential. The wallet may also obtain this information through other means (e.g. a verifiable data registry).
  2. (OPTIONAL) If the issuer expects certain credentials to be presented in the issuance flow, it requests the user to select and confirm presentation of those credentials.
  3. If the user has confirmed the presentation of certain credentials with the issuance request, the wallet prepares the process by obtaining a nonce from the issuer. This nonce will be used to prevent malicious wallets from being able to replay those presentations.
  4. The wallet sends an authorization request to the issuer, which determines the types of verifiable credentials the wallet (on behalf of the user) wants to obtain and may

also contain verifiable presentations if required by the issuer.



```
+--------------+   +----------+                                   +-------------+
| User         |   |  Wallet  |                                   |   Issuer    |
+--------------+   +----------+                                   +-------------+
        |                 |                                                |
        |    interacts    |                                                |
        |---------------->|                                                |
        |                 |  (1) [opt] obtain credential manifest          |
        |                 |----------------------------------------------->|
        |                 |            credential manifest                 |
        |                 |<-----------------------------------------------|
        |                 |                                                |
     (2) [opt] User selects credentials                                    |
        |                 |                                                |
        |                 |  (3) [opt] request presentation nonce          |
        |                 |----------------------------------------------->|
        |                 |      presentation nonce                        |
        |                 |<-----------------------------------------------|
        |                 |                                                |
        |                 |  (4) authorization req (claims, [opt] input, etc. )  |
        |                 |----------------------------------------------->|
        |                 |                                                |
        |      (4.1) User Login                                            |
        |                 |                                                |
        |                 |  (4.2) [opt] request additional VCs (OIDC4VP)  |
        |                 |<-----------------------------------------------|
        |                 |                                                |
   (4.2.1) [opt] User selects credentials                                  |
        |                 |                                                |
        |                 |  (4.2.2) VCs in Verifiable Presentations       |
        |                 |----------------------------------------------->|
        |                 |                                                |
        |    (4.3) User consents to credential issuance                    |
        |                 |                                                |
        |                 |  (5) authorization res (code)                  |
        |                 |<-----------------------------------------------|
        |                 |                                                |
        |                 |  (6) token req (code)                          |
        |                 |----------------------------------------------->|
        |                 |      access_token, id_token                    |
        |                 |<-----------------------------------------------|
        |                 |                                                |
        |                 |  (7) credential req (access_token, proofs, ...)  |
        |                 |----------------------------------------------->|
        |                 |      credential req (credentials OR acceptance_token)|
        |                 |<-----------------------------------------------|
        |                 |                                                |
        |                 |  (8) [opt] poll_credentials (acceptance_token) |
        |                 |----------------------------------------------->|
        |                 |      credentials OR not_ready_yet              |
        |                 |<-----------------------------------------------|
```

Figure 17: Overview of the OIDC credential issuance protocol (taken directly from the draft)

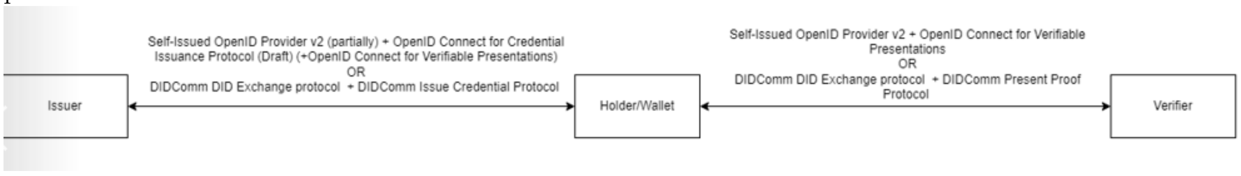
The wallet SHOULD use a pushed authorization request (see RFC9126) to first send the payload of the authorization request to the issuer and subsequently use the request_uri returned by the issuer in the authorization request. This ensures integrity and confidentiality of the request data and prevents any issues raised by URL length restrictions regarding the authorization request URL.

   (a) The issuer will typically authenticate the user in the first step of this process. For this purpose, the issuer might use a local or federated login, potentially informed by an id_token_hint, OR utilize, if present, verifiable presentations passed to the authorization request.
   (b) (OPTIONAL) The issuers MAY call back to the wallet to fetch verifiable credentials it needs as pre-requisite to issuing the requested credentials.
      – (CONDITIONAL) The wallet shows the content of the presentation request to the user. The user selects the appropriate credentials and consents.
      – (CONDITIONAL) The wallet responds with one or more verifiable presentations to the issuer.
   (c) The issuer asks the user for consent to issue the requested credentials.

5. The issuer responds with an authorization code to the wallet.
6. The wallet exchanges the authorization code for an Access Token and an ID Token.
7. The wallet uses this Access Token to request the issuance of the actual credentials. The types of credentials the wallet can request is limited to the types approved in the authorization request in (5). The credential request passes the key material the respective credential shall be bound to. If required by the issuer, the wallet also passes a proof of possession for the key material. This proof of possession uses the SHA256 hash of the Access Token as cryptographic nonce. This ensure replay protection of the proofs. The format of key material and proof of possession depends on the proof scheme and is expressed in a polymorphic manner on the protocol level.
   The Issuer will either directly respond with the credentials or issue an Acceptance Token, which is used by the wallet to poll for completion of the issuance process.
8. (OPTIONAL) The wallet polls the issuer to obtain the credentials previously requested in step (6). The issuer either responds with the credentials or HTTP status code "202" indicating that the issuance is not completed yet.

The draft also notes, that in the case "the issuer just wants to offer the user to retrieve an pre-existing credential, it can encode the parameter set of step (6) in a suitable representation and allow the wallet to start with step (6)" (e.g. encode the data in a QR code). Like the standard OpenID Connect SIOP specification, the draft also offers many detailed examples of all messages, however I will not include them in this report but rather point to the draft itself (`https://bitbucket.org/openid/connect/src/master/individual/draft-lodderstedt-openid-connect-4-credential-issuance-1_0.md`) in order to not bloat this report more than it already is.

To summarize, this illustration gives an overview of which protocols are used between which parties:


Self-Issued OpenID Provider v2 (partially) + OpenID Connect for Credential Issuance Protocol (Draft) (+OpenID Connect for Verifiable Presentations)
OR
DIDComm DID Exchange protocol + DIDComm Issue Credential Protocol
Self-Issued OpenID Provider v2 + OpenID Connect for Verifiable Presentations
OR
DIDComm DID Exchange protocol + DIDComm Present Proof Protocol
Issuer
Holder/Wallet
Verifier

7 Conclusions and Recommendations

During my internship I read and summarized a lot of different standards and gave considerations on many aspects relating to a potential eduID SSI. However, for SURF, it is important to know what to do with all this information and how they can proceed with the project of building their own SSI with eduID. This section will cover my conclusions and recommendations based on the data model and protocol standards described above.

7.1 Conclusions

7.1.1 Verifiable Credentials & DIDs

To start with, I worked heavily under the assumption that Verifiable Credentials and Decentralized Identifiers will be used. That is because these two data models are basically the standard at the very core within SSI environments and thus adopting these widely adopted standards is the most logical steps, especially when thinking in terms of interoperability. If everyone is using the same standards, it becomes easier for issuers and verifiers to join an SSI system, especially when they are already part of another SSI environment, since they do not need to adapt to a completely different data models. Verifiable Credentials in particular offer a lot of flexibility (which can also be a disadvantage in terms of interoperability, e.g. if different systems use vastly different credential schemas) and potential use cases. However, as I demonstrated earlier, the implementation is not always worked out in a standardized fashion yet. In this sense I would recommend observing the further path of the drafts I summarized earlier and the working groups within the W3C (Credentials Community Group) that are responsible for developing standardized implementation methods for Verifiable Credential concepts.
In terms of more advanced concepts and features defined in the VC specification, I highlighted the most useful ones (Zero-Knowledge Proofs, credential status, terms of use, refresh services, data schemas) earlier in this report (see section 6.1).
In regards to DIDs, I want to highlight the dilemma of choosing a DID method. There are a lot of methods to resolve DIDs, and there seems to be little to no consensus on which DID methods should be used or in which context which DID method should be used. This is a bit of a dilemma, since it makes interoperability a lot harder as you would need to implement loads of different DID methods to be able to resolve DIDs from different context which seems a tad bit excessive. So my recommendation here is to see which DID methods other parties use (Microsoft for example gave an overview of all standards and DID methods they used (see `https://linktr.ee/decentralizedidentity`)) and then make a decision based on that. Though what really needs to happen here is that the parties hosting/developing SSI systems and the DID Community need to come together and find a consensus to standardize the usage of specific methods in specific contexts.

7.1.2 Verifiable Data Registry

In the section on the role of the verifiable data registry during section 5.4, I listed the two main approaches that could be taken in regards to what kind of data structure should be used. These two approaches were to either use public infrastructure in the form of a public blockchain or host a database. Utilizing public infrastructure is obviously the better choice in terms of interoperability, with the problem of course being that everyone is using a different blockchain for their SSI system, defeating the entire interoperability aspect. In that sense, I would definitely at least to some research to at least explore some suitable options, like e.g. the ION blockchain Microsoft uses (`https://identity.foundation/ion/`).

A database would of course the less complicated option and, when looking at eduID as an isolated system, would also be a logical option, but would make interoperability at least more difficult. It is however difficult to give a recommendation at this point as to which data structure would be better suited for the purposes of eduID as that would require more research and considerations into these data structures, especially for blockchains.

7.1.3 Protocol Standards

Earlier I examined two SSI protocol standards in the form of DIDComm and OIDC SIOP. As I also mentioned earlier, both suffer a bit from a lack of widespread adoption and also somewhat from a lack of maturity in some parts, so giving a recommendation on which protocols to use is kind of difficult at the moment.
However, if I absolutely had to recommend one for now, I would recommend looking into the OpenID Connect SIOP protocols first. This is because Microsoft (who also played a big role in the development of these protocols) has adopted OIDC SIOP and because SURF is already using OpenID Connect in the context of SURFconext, so there is already some familiarity with the base OIDC protocol the SIOP extension are built on. Also, the specification for the OIDC SIOP extension are worked out a lot more in terms of message construction (including many examples) and considerations in regards to privacy and security, which also makes the implementation easier.
I think the most important aspect here I think is to be aligned with the most important other players within SSI, since if you use the same protocols as other widely used SSI systems, the burden of implementation is much lower for issuers and verifiers. The problem here is, that SSI as a concept is still very much at the beginning and not many people use SSI at all at the moment, so there are no real big SSI players yet. In that regard it is important to follow future developments in the field of SSI and which solutions will actually be used by the regular consumers.

7.1.4 On the topic of interoperability

Within this report I mentioned the terms "interoperability" and "interoperable" a lot. What this means in the context of eduID is that eduID can work together with other SSI implementations. This would for example means that credentials originally meant for the eduID ecosystem could also be used in Microsoft's SSI environment and vice versa. It also means that parties seeking to join an SSI ecosystem, like issuers and verifiers for example, do only need to implement one protocol standard and one data model standard in order to be able to join multiple different SSI ecosystems. Interoperability could thus be a major selling point for some potential issuers and verifiers to join the eduID ecosystem, since the barrier of entry would be lower for those that are already part of a different SSI environment.
Unfortunately, interoperability (or lack thereof) is currently the main problem in most SSI solution. Almost none of the current SSI implementations are truly interoperable with each other. The root of the problem is a lack of standardization in some areas and a lack of consensus on which standards to use in other areas. Finding such a consensus should be one of the main priorities within the SSI community for the next few years. SURF could also be active in that regard by participating in relevant working groups within standards organizations such as the W3C and trying to actively cooperate with other parties within SSI to proactively taking part in the process for standardization and achieving consensus. However, I want to note that it should be kept in mind that, for some of the bigger players, interoperability may be less of priority and some might even have an incentive not to push for standardization to push users to actually use their product and not someone else's.

Another thing to note here is that one needs to be careful with flexible standards such as Verifiable Credentials when interoperability is a desired property. If they are customised too much in a given implementation, one might end up in a situation similar to SAML where implementations are often vastly different from one another. This again highlights the importance of conventions and consensuses about the implementation of certain standards.

7.1.5 Necessary infrastructure

To actually implement an SSI system, you first need to set up the necessary infrastructure for SSI. This has to be done by all the parties involved, thus issuers need to set up their own infrastructure, verifiers need to set up their own infrastructure and the party providing SSI for users, SURF in the case of eduID, also needs to set-up infrastructure that the other participants can use. I will try to provide a rough overview of what kind of infrastructure each party needs to set up, grouped by role.

SURF
The most obvious piece of infrastructure that SURF needs to provide in order to turn eduID into a full-blown SSI and not just a credential for a different SSI system is of course the wallet app itself. This also means implementing all relevant protocols and possible checks into the wallet app that fit the policies set in place for eduID. SURF probably also needs a database where the general account data of eduID accounts is stored. This database probably already exists, so there is probably no need to set up another one for that purpose. Also, if SURF wants to allow for two classes of verifiers (trusted and "untrusted"), a database or list of trusted verifiers that the wallet can access to enforce certain policies needs to be provided as well.
If SURF chooses to host their own verifiable data registry instead of using public infrastructure, then obviously that needs to be set up and hosted by SURF as well. But even if SURF chooses to utilize public infrastructure (e.g. in the form a public, permissionless blockchain), that infrastructure also needs to be properly integrated into some protocols and the software used by all parties, as issuers, verifiers and the wallet app need to interact with it to look up or store identifiers and credential schemas.
SURF could potentially also provide software to issuers and verifiers so that they just have to set up the setup on their infrastructure instead of having them implement their own software. However this is by no means a must and there might be reasons to not provide them with their own software. These reasons would be that the other parties either already have SSI software that is compatible with eduID (e.g. because it uses the protocols and data models), want to save up on costs/time or want to have more customization in some parts (though that could also be possible when the software is provided by SURF, e.g. as an open-source software with a public Github repository). Reasons to provide software for SURF would be that it would make integration of parties into the ecosystem easier, could still allow for some customization and could be used to enforce specific eduID policies also on the side of issuers and verifiers (server side) and not only on the client side (wallet app).
Yet another option would be for SURF to offer verification/issuance as a service that handles the actual SSI part of the communication (i.e. the protocols and data models) like SURF offers federated log-in as a service with SURFconext. This approach would make it much easier to actually enforce eduID specific policies and also of course makes it much easier for issuers and verifiers to join the eduID ecosystem as they could just use the service. This would of course require an additional level of trust in SURF on the part of issuers and verifiers.

Issuers

Issuers at the very least need to set up some endpoints to be reached by holders (whether this is in the form of separate servers or on existing servers is not relevant for now). They also need to adapt their services to integrate SSI functionalities, by generating QR codes/deep links at the appropriate locations. They also need to set up specific software for SSI to implement the needed protocols and be able to generate verifiable credentials and the necessary proofs. This software could either be provided by SURF or they could design and implement their own software. Depending on the issuer, they may also need to be able to verify verifiable presentations (e.g. when users don't use some form of log-in to authenticate themselves before issuance). They also need to be able to set up a public DID as their public identifier and possibly also generate and keep track of private identifiers, depending on the protocols used.

Verifiers
Like issuers, verifiers also need to set up appropriate endpoints where they can be reached by holders for SSI-related interactions. They also need to set up specific software to implement the needed protocols and be able to properly request and verify verifiable presentation. If they want to store certain claims, then they also need some sort of database to store the data together with the associated identifier of the eduID user or the account the eduID user is using on their service. They also need to properly integrate the SSI functionality into their services/websites. Again, just like issuer, they also need to be able to set up a public DID as their public identifier and possibly also generate and keep track of private identifiers, depending on the protocols used. So issuers and verifiers need to set up very similar infrastructure.

7.2 Recommendations

After taking everything I have written up until now into consideration, there is only one question that remains: What are the next steps and which aspects need to be researched and/or tested further? Of course, eduID SSI is still at the very beginning so there is still to a lot to do. I want to give some pointers here on where to start and try to give an overview of things that can be done in the near future.

7.2.1 Experimentation/Testing

I think the first step in trying to build an eduID SSI wallet should be to get familiarized with the core underlying data models, i.e. verifiable credentials and decentralized identifiers. To be more concrete, here is a list of aspects that should be examined and tested in a pilot or proof-of-concept:

- Test how to construct a verifiable credential. Maybe start with designing a few schemas that could be used in practice (e.g. a schema for diplomas or a schema for a student credential). For this purpose inspiration may be taken from existing schemas, but it might also make sense to already try to either find or create a schema that could actually be used in a potential eduID ecosystem. In the same vain, it would also be necessary to a context file to define the necessary terms (for the @context parameter of the VCs). After that, try to design some credentials of that kind, starting with the most basic features, i.e. basic metadata variables, some basic claims and a proof, and later also testing the more advanced concepts. This also means examining how to construct viable proofs.

- Test how to create verifiable presentations from verifiable credentials and construct the necessary proofs. Here all kinds of possibilities should be examined, e.g. presentation containing some claims from one credential without encompassing the whole credential, presentations containing derived claims, presentations containing claims from multiple different credentials.

- Test how to construct a DID and its corresponding document.

After getting familiarized with the basic data models, the next aspects that need familiarizing are the interactions between the different parties, i.e. the communication protocols. For this purpose, one could build two test clients, one simulating an issuer and a verifier and one simulating a wallet, both with very limited functionality, to test the protocols, the construction of requests and messages and the verification of proofs.

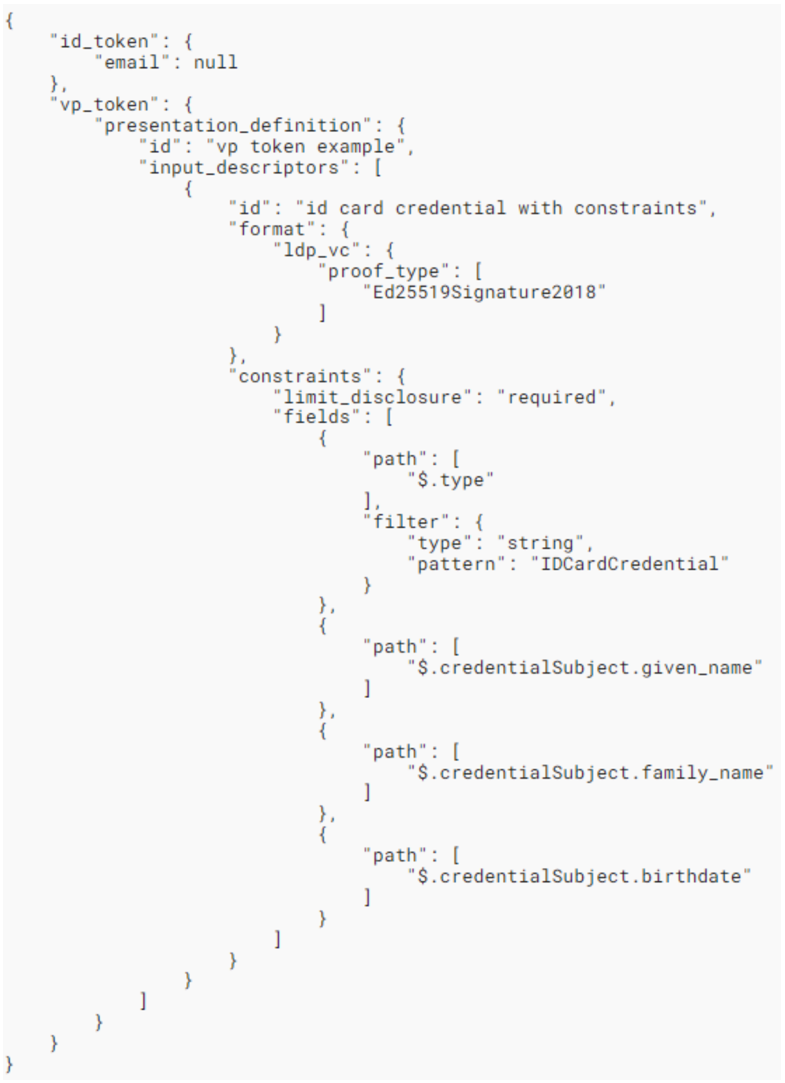
7.2.2 Discussion points & further research

Some aspects need further discussion within the eduID team and/or further research. For the sake of this report I will list the most important ones here:

- Which data structure should be used to fill the role of the verifiable data registry?
- Identifying potential participants within a possible eduID trust framework (issuers and "trusted" verifiers)
- How to address the misuse/abuse cases where the issue is not addressed by the used data models and protocols already?
- What should an eduID trust framework look like in practice? What should contracts with other parties look like? What responsibilities should be placed on contractual partners?
- Get legal advice from qualified party on all relevant aspects (e.g. management of private data and identifiers, legal requirements for trust framework, relevant legislation)
- How could rules be enforced on a technical and organizational level?
- If storage/consent receipts should be used, how should they work and how would they need to be implemented?

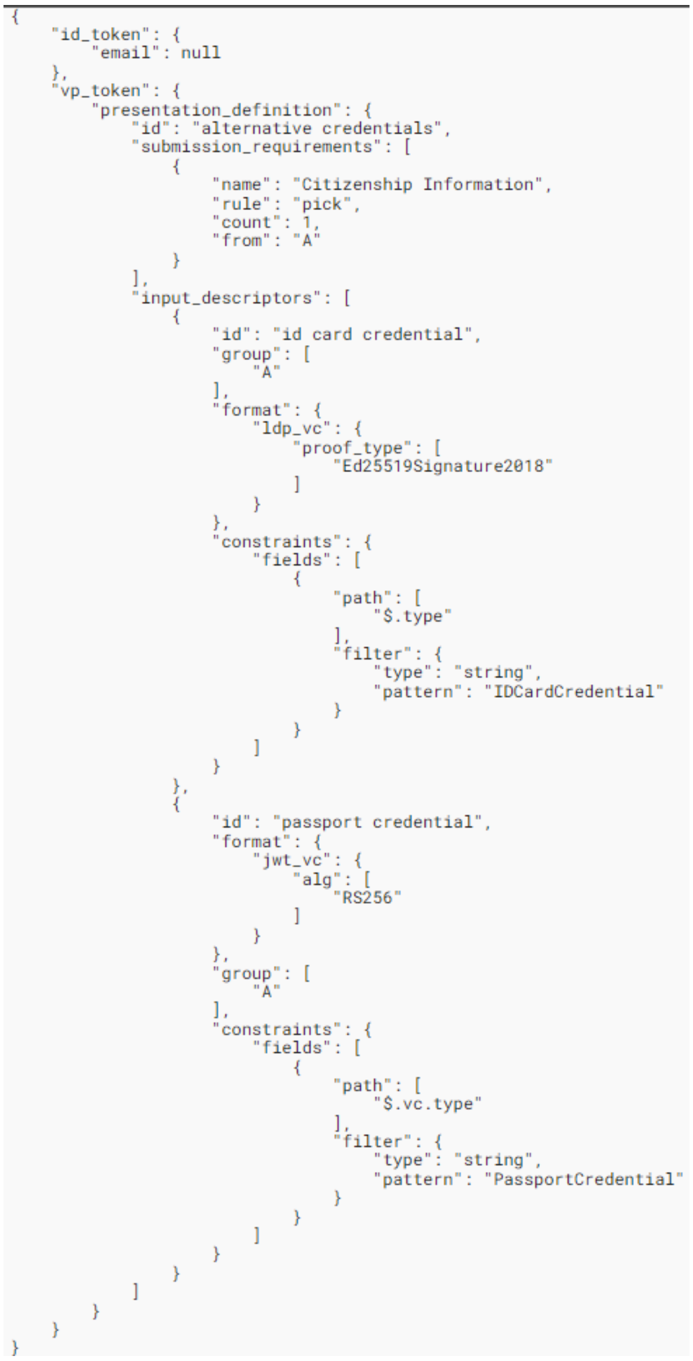
A Additional Examples

A.1 Self-Issued OpenID Provider Authentication Request



```
{
    "id_token": {
        "email": null
    },
    "vp_token": {
        "presentation_definition": {
            "id": "vp token example",
            "input_descriptors": [
                {
                    "id": "id card credential with constraints",
                    "format": {
                        "ldp_vc": {
                            "proof_type": [
                                "Ed25519Signature2018"
                            ]
                        }
                    },
                    "constraints": {
                        "limit_disclosure": "required",
                        "fields": [
                            {
                                "path": [
                                    "$.type"
                                ],
                                "filter": {
                                    "type": "string",
                                    "pattern": "IDCardCredential"
                                }
                            },
                            {
                                "path": [
                                    "$.credentialSubject.given_name"
                                ]
                            },
                            {
                                "path": [
                                    "$.credentialSubject.family_name"
                                ]
                            },
                            {
                                "path": [
                                    "$.credentialSubject.birthdate"
                                ]
                            }
                        ]
                    }
                }
            ]
        }
    }
}
```

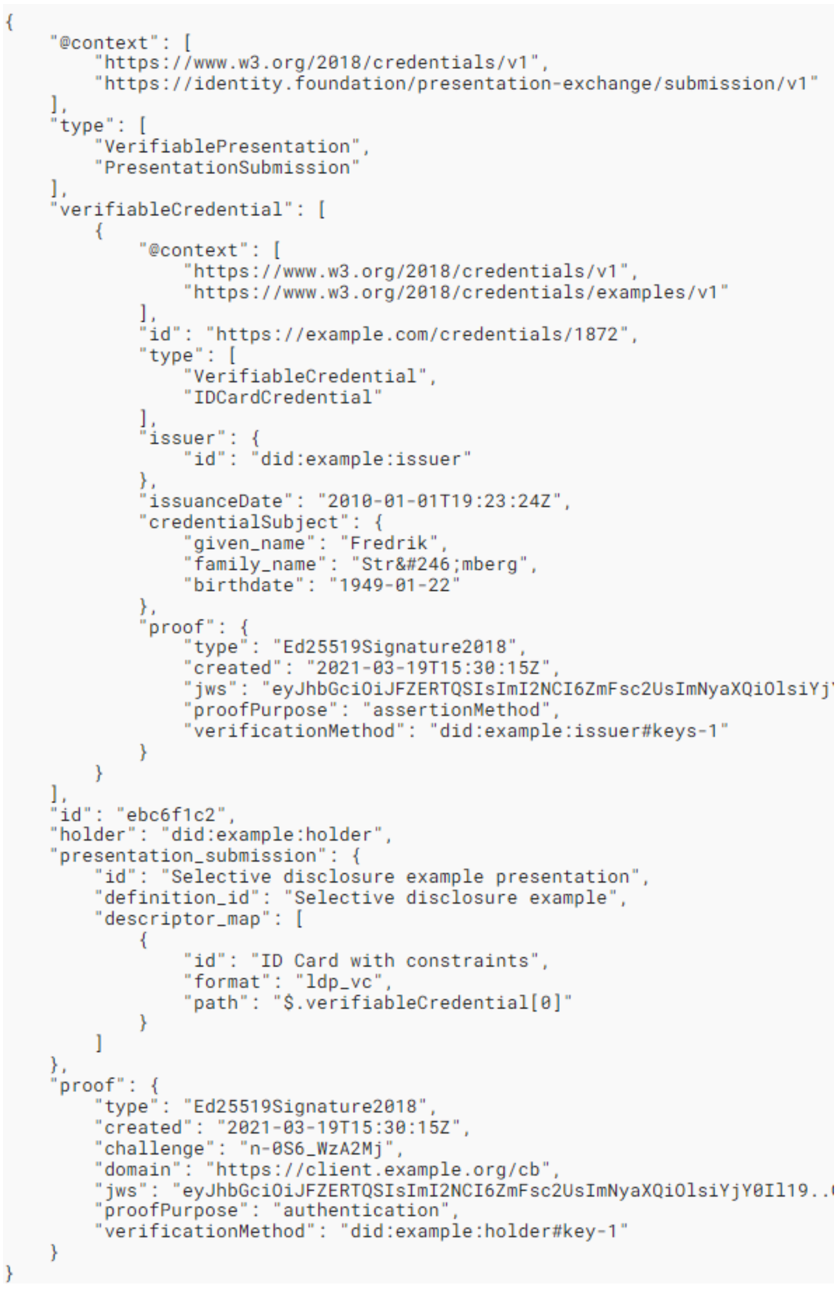
Figure 18: Example "claims" parameter where the RP requests selective disclosure or certain claims from a credential of a particular type



```
{
    "id_token": {
        "email": null
    },
    "vp_token": {
        "presentation_definition": {
            "id": "alternative credentials",
            "submission_requirements": [
                {
                    "name": "Citizenship Information",
                    "rule": "pick",
                    "count": 1,
                    "from": "A"
                }
            ],
            "input_descriptors": [
                {
                    "id": "id card credential",
                    "group": [
                        "A"
                    ],
                    "format": {
                        "ldp_vc": {
                            "proof_type": [
                                "Ed25519Signature2018"
                            ]
                        }
                    },
                    "constraints": {
                        "fields": [
                            {
                                "path": [
                                    "$.type"
                                ],
                                "filter": {
                                    "type": "string",
                                    "pattern": "IDCardCredential"
                                }
                            }
                        ]
                    }
                },
                {
                    "id": "passport credential",
                    "format": {
                        "jwt_vc": {
                            "alg": [
                                "RS256"
                            ]
                        }
                    },
                    "group": [
                        "A"
                    ],
                    "constraints": {
                        "fields": [
                            {
                                "path": [
                                    "$.vc.type"
                                ],
                                "filter": {
                                    "type": "string",
                                    "pattern": "PassportCredential"
                                }
                            }
                        ]
                    }
                }
            ]
        }
    }
}
```

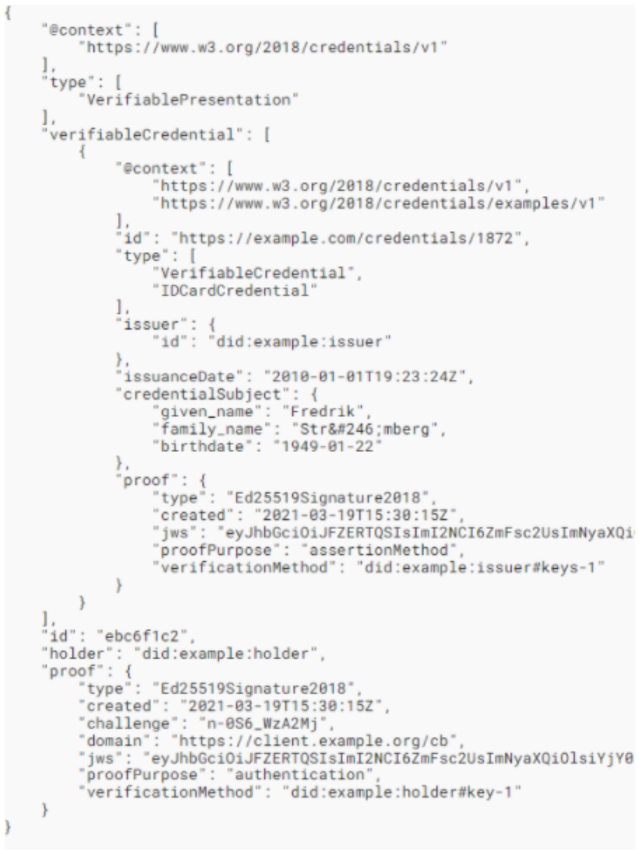
Figure 19: Example "claims" parameter where RP also asks for alternative credentials being presented

A.2 Self-Issued OpenID Provider Authentication Response

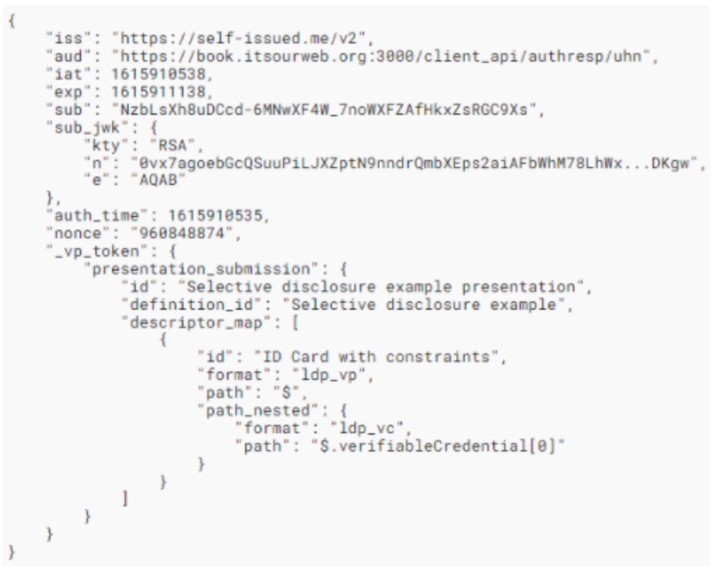


```
{
    "@context": [
        "https://www.w3.org/2018/credentials/v1",
        "https://identity.foundation/presentation-exchange/submission/v1"
    ],
    "type": [
        "VerifiablePresentation",
        "PresentationSubmission"
    ],
    "verifiableCredential": [
        {
            "@context": [
                "https://www.w3.org/2018/credentials/v1",
                "https://www.w3.org/2018/credentials/examples/v1"
            ],
            "id": "https://example.com/credentials/1872",
            "type": [
                "VerifiableCredential",
                "IDCardCredential"
            ],
            "issuer": {
                "id": "did:example:issuer"
            },
            "issuanceDate": "2010-01-01T19:23:24Z",
            "credentialSubject": {
                "given_name": "Fredrik",
                "family_name": "Str&#246;mberg",
                "birthdate": "1949-01-22"
            },
            "proof": {
                "type": "Ed25519Signature2018",
                "created": "2021-03-19T15:30:15Z",
                "jws": "eyJhbGciOiJFZERTQSIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYj
                "proofPurpose": "assertionMethod",
                "verificationMethod": "did:example:issuer#keys-1"
            }
        }
    ],
    "id": "ebc6f1c2",
    "holder": "did:example:holder",
    "presentation_submission": {
        "id": "Selective disclosure example presentation",
        "definition_id": "Selective disclosure example",
        "descriptor_map": [
            {
                "id": "ID Card with constraints",
                "format": "ldp_vc",
                "path": "$.verifiableCredential[0]"
            }
        ]
    },
    "proof": {
        "type": "Ed25519Signature2018",
        "created": "2021-03-19T15:30:15Z",
        "challenge": "n-0S6_WzA2Mj",
        "domain": "https://client.example.org/cb",
        "jws": "eyJhbGciOiJFZERTQSIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19..
        "proofPurpose": "authentication",
        "verificationMethod": "did:example:holder#key-1"
    }
}
```

Figure 20: Example of a vp_token in a Self-Issued OpenID Provider Authentication Response

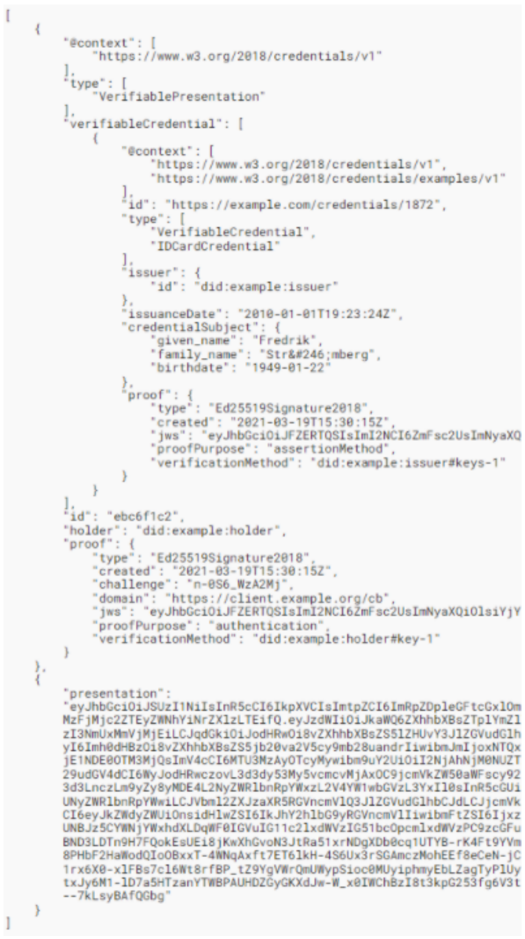


```
{
    "@context": [
        "https://www.w3.org/2018/credentials/v1"
    ],
    "type": [
        "VerifiablePresentation"
    ],
    "verifiableCredential": [
        {
            "@context": [
                "https://www.w3.org/2018/credentials/v1",
                "https://www.w3.org/2018/credentials/examples/v1"
            ],
            "id": "https://example.com/credentials/1872",
            "type": [
                "VerifiableCredential",
                "IDCardCredential"
            ],
            "issuer": {
                "id": "did:example:issuer"
            },
            "issuanceDate": "2010-01-01T19:23:24Z",
            "credentialSubject": {
                "given_name": "Fredrik",
                "family_name": "Str&#246;mberg",
                "birthdate": "1949-01-22"
            },
            "proof": {
                "type": "Ed25519Signature2018",
                "created": "2021-03-19T15:30:15Z",
                "jws": "eyJhbGciOiJFZERTQSIsImI2NCI6ZmFsc2UsImNyaXQi
                "proofPurpose": "assertionMethod",
                "verificationMethod": "did:example:issuer#keys-1"
            }
        }
    ],
    "id": "ebc6f1c2",
    "holder": "did:example:holder",
    "proof": {
        "type": "Ed25519Signature2018",
        "created": "2021-03-19T15:30:15Z",
        "challenge": "n-0S6_WzA2Mj",
        "domain": "https://client.example.org/cb",
        "jws": "eyJhbGciOiJFZERTQSIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0
        "proofPurpose": "authentication",
        "verificationMethod": "did:example:holder#key-1"
    }
}
```

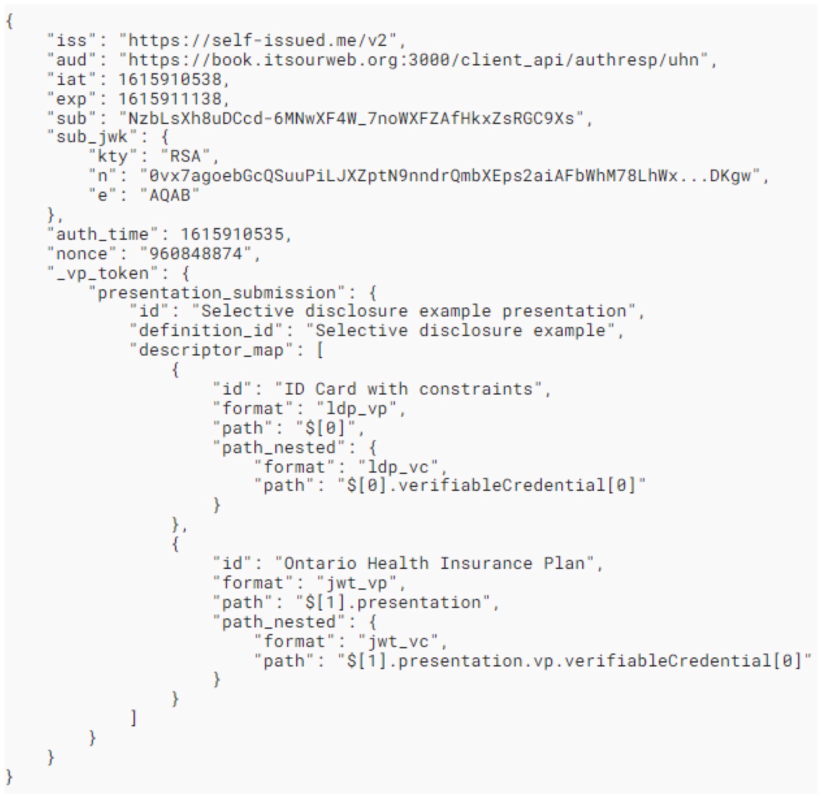


```
{
    "iss": "https://self-issued.me/v2",
    "aud": "https://book.itsourweb.org:3000/client_api/authresp/uhn",
    "iat": 1615910538,
    "exp": 1615911138,
    "sub": "NzbLsXh8uDCcd-6MNwXF4W_7noWXFZAfHkxZsRGC9Xs",
    "sub_jwk": {
        "kty": "RSA",
        "n": "0vx7agoebGcQSuuPiLJXZptN9nndrQmbXEps2aiAFbWhM78LhWx...DKgw",
        "e": "AQAB"
    },
    "auth_time": 1615910535,
    "nonce": "960848874",
    "_vp_token": {
        "presentation_submission": {
            "id": "Selective disclosure example presentation",
            "definition_id": "Selective disclosure example",
            "descriptor_map": [
                {
                    "id": "ID Card with constraints",
                    "format": "ldp_vp",
                    "path": "$",
                    "path_nested": {
                        "format": "ldp_vc",
                        "path": "$.verifiableCredential[0]"
                    }
                }
            ]
        }
    }
}
```

Figure 21: Example of a vp_token containing a single verifiable presentation (left image) and the _vp_token in the corresponding id_token (right image)



```
[
    {
        "@context": [
            "https://www.w3.org/2018/credentials/v1"
        ],
        "type": [
            "VerifiablePresentation"
        ],
        "verifiableCredential": [
            {
                "@context": [
                    "https://www.w3.org/2018/credentials/v1",
                    "https://www.w3.org/2018/credentials/examples/v1"
                ],
                "id": "https://example.com/credentials/1872",
                "type": [
                    "VerifiableCredential",
                    "IDCardCredential"
                ],
                "issuer": {
                    "id": "did:example:issuer"
                },
                "issuanceDate": "2010-01-01T19:23:24Z",
                "credentialSubject": {
                    "given_name": "Fredrik",
                    "family_name": "Str&#246;mberg",
                    "birthdate": "1949-01-22"
                },
                "proof": {
                    "type": "Ed25519Signature2018",
                    "created": "2021-03-19T15:30:15Z",
                    "jws": "eyJhbGciOiJFZERTQSIsImI2NCI6ZmFsc2UsImNyaXQ
                    "proofPurpose": "assertionMethod",
                    "verificationMethod": "did:example:issuer#keys-1"
                }
            }
        ],
        "id": "ebc6f1c2",
        "holder": "did:example:holder",
        "proof": {
            "type": "Ed25519Signature2018",
            "created": "2021-03-19T15:30:15Z",
            "challenge": "n-0S6_WzA2Mj",
            "domain": "https://client.example.org/cb",
            "jws": "eyJhbGciOiJFZERTQSIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY
            "proofPurpose": "authentication",
            "verificationMethod": "did:example:holder#key-1"
        }
    },
    {
        "presentation":
        "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6ImRpZDpleGFtcGxlOm
        MzFjMjc2ZTEyZWNhYiNrZXlzLTEifQ.eyJzdWIiOiJkaWQ6ZXhhbXBsZTplYmZl
        zI3NmUxMmVjMjEiLCJqdGkiOiJodHRwOi8vZXhhbXBsZS5lZHUvY3JlZGVudGlh
        yI6Imh0dHBzOi8vZXhhbXBsZS5jb20va2V5cy9mb28uandrIiwibmJmIjoxNTQx
        jE1NDE0OTM3MjQsImV4cCI6MTU3MzAyOTcyMywibm9uY2UiOiI2NjAhNjM0NUZT
        29udGV4dCI6WyJodHRwczovL3d3dy53My5vcmcvMjAxOC9jcmVkZW50aWFscy92
        3d3LnczLm9yZy8yMDE4L2NyZWRlbnRpYWxzL2V4YW1wbGVzL3YxIl0sInR5cGUi
        UNyZWRlbnRpYWwiLCJVbml2ZXJzaXR5RGVncmVlQ3JlZGVudGlhbCJdLCJjcmVk
        CI6eyJkZWdyZWUiOnsidHlwZSI6IkJhY2hlbG9yRGVncmVlIiwibmFtZSI6Ijxz
        UNBJz5CYWNjYWxhdXLDqWF0IGVuIG11c2lxdWVzIG51bcOpcmlxdWVzPC9zcGFu
        BND3LDTn9H7FQokEsUEi8jKwXhGvoN3JtRa51xrNDgXDb0cq1UTYB-rK4Ft9YVm
        8PHbF2HaWodQIoOBxxT-4WNqAxft7ET6lkH-4S6Ux3rSGAmczMohEEf8eCeN-jC
        1rx6X0-xlFBs7cl6Wt8rfBP_tZ9YgVWrQmUWypSioc0MUyiphmyEbLZagTyPlUy
        txJy6M1-lD7a5HTzanYTWBPAUHDZGyGKXdJw-W_x0IWChBzI8t3kpG253fg6V3t
        --7kLsyBAfQGbg"
    }
]
```



```
{
    "iss": "https://self-issued.me/v2",
    "aud": "https://book.itsourweb.org:3000/client_api/authresp/uhn",
    "iat": 1615910538,
    "exp": 1615911138,
    "sub": "NzbLsXh8uDCcd-6MNwXF4W_7noWXFZAfHkxZsRGC9Xs",
    "sub_jwk": {
        "kty": "RSA",
        "n": "0vx7agoebGcQSuuPiLJXZptN9nndrQmbXEps2aiAFbWhM78LhWx...DKgw",
        "e": "AQAB"
    },
    "auth_time": 1615910535,
    "nonce": "960848874",
    "_vp_token": {
        "presentation_submission": {
            "id": "Selective disclosure example presentation",
            "definition_id": "Selective disclosure example",
            "descriptor_map": [
                {
                    "id": "ID Card with constraints",
                    "format": "ldp_vp",
                    "path": "$[0]",
                    "path_nested": {
                        "format": "ldp_vc",
                        "path": "$[0].verifiableCredential[0]"
                    }
                },
                {
                    "id": "Ontario Health Insurance Plan",
                    "format": "jwt_vp",
                    "path": "$[1].presentation",
                    "path_nested": {
                        "format": "jwt_vc",
                        "path": "$[1].presentation.vp.verifiableCredential[0]"
                    }
                }
            ]
        }
    }
}
```

Figure 22: Example of a vp_token containing multiple verifiable presentations (left image) and the _vp_token in the corresponding id_token (right image)