Master's thesis

Master's Programme in Computer Science

On Self-Sovereign Identity: Verifiable Credentials and Presentations with OpenID Connect

Miika Pärni

June 8, 2023

Faculty of Science

University of Helsinki

Contact information

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki,Finland

Email address: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Faculty of Science	Master's Programme in Computer Science

Tekijä — Författare — Author

Miika Pärni

Työn nimi — Arbetets titel — Title

On Self-Sovereign Identity: Verifiable Credentials and Presentations with OpenID Connect

Ohjaajat — Handledare — Supervisors

Prof. Valtteri Niemi

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's thesis	June 8, 2023	65 pages

Tiivistelmä — Referat — Abstract

Self-Sovereign Identity is a new concept of managaging digital identities in the digital services. The purpose of the Self-Sovereign Identity is to place the user in the center and move towards decentralized model of identity management. Verifiable Credentials, Verifiable Presentations, Identity Wallets and Decentralized Identifiers are part of the Self-Sovereign Identity model. They have also been recently included in the OpenID Connect specifications to be used with the widely used authentication layer built on OAuth 2.0. The OpenID Connect authentication can now be leveraged with the Decetralized Identifiers (DIDs) and the public keys contained in DID Documents.

This work assessed the feasibility of integrating the Verifiable Credentials, Verifiable Presentations and Decentralized Identifiers with OpenID Connect in the context of two use cases. The first use case is to integrate the Verifiable Credentials and Presentations into an OpenID Connect server and utilise Single Sign-On in federated environment. The second use case is to ignore the OpenID Provider and enable the Relying Party to authenticate directly with the Identity Wallet. Custom software components, the Relying Party, the Identity Wallet and the Verifiable Credential Issuer were built to support the assessments.

Two new authorization flows were designed for the two use cases. The Federated Verifiable Presentation Flow describes the protocol of Relying Party authenticating with OpenID Provider which receives the user information from the Wallet. The flow does not require any changes for any Relying Party using the same OpenID Provider to authenticate and utilise Single Sign-On. The Verifiable Presentation Flow enables the Relying Party to authenticate directly with the Wallet. However, this flow requires multiple changes to Relying Party and benefits of federated environment are not available, e.g., the Single Sign-On. Both of the flows are useful for their own specific use cases. The new flows are utilising the new segments of the Self-Sovereign Identity and are promising steps towards self-sovereignty.

ACM Computing Classification System (CCS)

Security and privacy → Human and societal aspects of security and privacy → Usability in security and privacy

Avainsanat — Nyckelord — Keywords

self-sovereign identity, verifiable credentials, decentralized identifiers, security, privacy, blockchain

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Networking study track

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Faculty of Science	Master's Programme in Computer Science

Tekijä — Författare — Author

Miika Pärni

Työn nimi — Arbetets titel — Title

On Self-Sovereign Identity: Verifiable Credentials and Presentations with OpenID Connect

Ohjaajat — Handledare — Supervisors

Prof. Valtteri Niemi

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's thesis	8.6.2023	65 pages

Tiivistelmä — Referat — Abstract

Itsehallittava identiteetti on uusi yksilökeskeinen digitaalisten identiteettien ja palveluiden hallitsemiseen sekä käyttöön tarkoitettu viitekehys. Viitekehyksen olennaisia sovelluksia ja tietorakenteita ovat todennettavat käyttäjäattribuutit, attribuuttitodistukset, identiteettilompakot ja hajautetut tunnisteet. Paljon käytettyyn tunnistautumisprotokollaan, OpenID Connect:iin on lisätty tuki ja määrittelyt todennettaville käyttäjäattribuuteille, attribuuttitodistuksille sekä hajautetuille tunnisteille. Hajautettujen tunnisteiden tuottamia tunnistedokumentteja voidaan käyttää julkisten avainten säilytykseen ja niihin sopivien digitaalisten allekirjoitusten todentamiseen.

Tämä toteutettavuustutkimus arvioi kahden käyttötapauksen valossa, kuinka käytännöllistä on yhdistää todennettavat käyttäjäattribuutit, attribuuttitodistukset ja hajautetut tunnisteet OpenID Connect -protokollaan. Ensimmäinen tarkasteltava tapaus käsittää näiden uusien tietorakenteiden integroinnin keskitetyssä, OpenID Connect:ia ilmentävässä identiteetin- ja pääsynhallinnan järjestelmässä. Keskitetty järjestelmä käyttää federoitua autentikointitapaa identiteettilompakkoa vasten. Näin kertakirjautumistoiminnallisuutta voidaan hyödyntää eri palveluntarjoajien sovellusten välillä. Toisessa tapauksessa uusia tietorakenteita hyödynnettiin suoraan palveluntarjoajan ja identiteettilompakon välillä.

Tutkittavia tapauksia varten luotiin kaksi uutta autorisointimenetelmää. Federoidussa attribuuttitodistusmenetelmässä OpenID-palvelin saa käyttäjätiedot identiteettilompakolta, luo kertakirjautmisistunnon ja välittää käyttäjätiedot edelleen palveluntarjoajasovellukselle. Tämä menetelmä vaatii muutoksia ainoastaan OpenID-palvelimeen. Attribuuttitodistusmenetelmässä palveluntarjojasovellus saa käyttäjätiedot suoraan identiteettilompakolta, mutta kertakirjautumista ei tällöin voida hyödyntää ja palveluntarjoja joutuu tekemään useita muutoksia omaan järjestelmäänsä. Molemmille menetelmille löytyy hyödyllisiä käyttötapauksia ja ne soveltavat uusia, lupaavan itsehallittavan identiteettimallin osa-alueita.

ACM Computing Classification System (CCS)

Security and privacy → Human and societal aspects of security and privacy → Usability in security and privacy

Avainsanat — Nyckelord — Keywords

self-sovereign identity, verifiable credentials, decentralized identifiers, security, privacy, blockchain

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Networking study track

Contents

1 Introduction

Digital identities and their management have become ubiquitous all over the Internet. As the services have developed in complexity, connectivity and availability, the methods to provide identity and access management have also evolved over the years. People are used to do shopping, file tax reports, enroll to universities, apply passports, use social media and banking services in different online applications.

Many of the modern applications require some profiling of the user. The application must know something about the user, e.g., the age, name, date of birth or social security number. The banks in the European Union are required to know their customers. Thus the information required about the user is extensive, including, the real identity of the user has to be verified.

Then there are services which require less information before their application can be used. However, sometimes these services gather information about the user by tracking the users' interests and activity within the application. Cases have been reported [1] where some social media services have been selling their users' information to external third parties without requesting the permission for sharing the data from their owners. Information gathering is exercised by many actors from individuals to business, and all the way to state level actors.

It is important to notice that the information gathered by the services does not usually mean just collecting the least amount of user data which is required for the service to be functional. At times the services request more user data than what is necessary. Sometimes gathering extensive information is inconspicuous and sometimes for some parties, it is profitable business.

Self-Sovereign Identity is a new and different concept for managing digital identities compared to earlier models, e.g., the federated identity model. User could login or register in to services they already have used to. The applications are required to request the credentials they need and user can either approve or reject those requests. The credentials can be issued by multiple different providers including the user themself. It is up to service providers' policies, on which issuer they trust. The service providers do not typically rely on a single identity provider. It is also possible to use the features of Self-Sovereign Identity in federated environment.

User is in control of their multiple different credentials and these are stored digitally on device such as smartphone or in a cloud application. The credentials are, e.g., driver's licence, identification card, grocery store customer membership card or university degree certificate. In case a digital service would require some attribute of the user's identity, for example, age, the service would have to ask the permission from the user. The user can then decide whether to release such information or not. One extension to the previous models, is that the user attribute requirement can be even more fine grained. A service could have a requirement of at least 18 years of age and the exact date of birth is irrelevant. In this case the service would get assured information, from a trusted third party, that the user is at least 18 years old.

Following use case describes the example above step by step:

1. The user registers in to service, online streaming service, and gets notified that the service requires to know if the user is at least 18 years old. User can pick which of their credentials they would use to prove that. For this service, the driver's licence is enough to prove the age.

2. The user can inspect where the credential request is coming from and decide whether to grant access to the requested information of age or not.

3. In the case the user allows the information for the the service, the service then validates and confirms the required information. The authenticity is verified by the digital signature of the granted driver's licence and is also verified that the driver's licence has been issued by a trusted third party.

The service receives only relevant amount of information about the age, an assurance from a trusted source, that the age is minimum of 18 years. The user required to release only a piece of information compared to giving the full date of birth. This example highlights the concept of minimal closure of information. There could be other similar types of requirements, e.g., does this person have certain type of driver's licence, does this person have certain club membership or has this person had certain infection for the past few weeks.

2 Background

Managing digital identities, authentication and authorization have gone through various changes over the years. Common entities to the identity models are the service providers (SP) and identity providers (IdP). The SPs provides the service or the application for the user. The IdPs provides the user profile information to SPs. The user profiles contain various identifying attributes of the user. Each period has had its prevalent identity model and these models are described from oldest to the most recent:

1. *The Isolated User Identity (SILO)* [2], the baseline identity management model. This centralized model involves only the end user and the service provider (SP) with its identity provider (IdP) and these can be regarded analogous in this model. These parties operate only in a single identity domain and the identities in that domain are not valid in other service providers' domains. Intranet of small company is an example of SILO model. The employees and employer can use the company's internal resources and services. The company has its own users and identity management system and services behind it. Big companies also use this model in some services, e.g., one can log in to Google mail but that login does not grant access to Yahoo mail.

2. *Federated Model* [3]. This model contains more complexity, flexibility and usability than the SILO model. In this model, there is typically one central IdP which manages the identities and the identity attributes. The end user can visit multiple different SPs while authenticating only once in the IdP. There is a strong trust relationship between the SP and the IdP. The connections are configured beforehand, e.g, a network of universities, where end user could log in to single IdP, browse and enroll to courses on different universities. This common feature of the federated model is called *Single sign-on*. Typically Security Assertion Markup Language (SAML) [4] is used in implementations of this model.

3. *User-centric Model*. This model usually combines technologies such as OpenID Connect [5] and OAuth 2.0 [6] and works like federated model, except that there is no need to exchange certificates for common use cases between SP or the IdP. The OpenID Connect specification [5] also allows dynamic client registration, i.e., reg-

istering RPs on demand for utilising the *Single sign-on* feature. In this model the Identity Provider (IdP) is commonly known as OpenID Provider (OP) and the service provider as Relying Party (RP). Like federated model, the user-centric model is also common. The user-centric model is encountered in use cases seen over the Internet, such as 'Sign In With Google', 'Facebook login' or 'Sign in with Apple'. The user-centric and federated models seem disjoint, but in practice many features overlap. User-centric model is also commonly used in federated authentication.

4. *Self-Sovereign Identity* is a relatively new decentralized model for distributing and verifying information of digital identities on the Internet. Many sources are pointing to a web article by Christopher Allen in the year 2016 [7] as the starting point. The Self-Sovereign Identity has since gained popularity in academia as well. There are multiple studies and surveys around the topic, for example, thorough overview of the system and components [8] [9] and also with formal definition of the different characteristics [3].

The main focus of Self-Sovereign Identity model is user-centric. User control of their digital identities, credentials and attributes is in the core of Self-Sovereign Identity. The user decides when, to whom and what kind of data should be released at a given occasion. This works against the current trend of various large companies, utilising combination of federated and user-centric identity models, having profiles of people with varying information and exchanging that information with other parties without the user's consent.

It is to be noted that the data could be a small subset of one's partial identity, such as a simple answer to whether the user is over 18 years of age or not. If that answer would be sufficient for the service to work, there is no need to ask for full date of birth.

One of the basic use cases would involve three parties [8]:

1. *User* who controls the identity and the claims within. These claims can be expressions about the user or they can be cryptographically verifiable ones issued by a trusted third party (TTP).

2. *Issuer* who issues the claims which are the building blocks of partial identity and the subjects the relying parties are interested of. The issuer might be a trusted governmental body, authorized to issue passports, therefore trusted to prove one's age.

3. *Relying party or Verifier*, acting as a service provider who consumes the partial

identity on user's permission. Permission is asked for those claims the Relying Party has defined in its policies. These claims or attributes required by the policies are also known as the scope. Relying Party is able to verify the authenticity of the claims which are cryptographically signed.

The model itself is also suitable for such as a scenario where end-user is able to handle tasks on behalf of their company which would involve authentication, permissions and several verifications.

User stores the claims as verifiable credentials securely in digital wallets. Digital wallets are applications in users' smart phones or software run in cloud. Digital wallets are a concept similar to real world counterparts, but the digital wallets can store more credentials than one could fit into real wallet. Digital wallets would work online and offline, could be restored in case of lost device and the data could be transferred from one wallet provider's solution to another.

Essential components and terminology of the Self-Sovereign Identity are introduced in the following sections.

Identity and Partial Identity

Identity in the online world is called digital identity [10] or partial identity [3]. Digital identity might not have anything to do with the real world identity. In contrast, digital identity can also include multiple personally identifiable attributes and then the link to real identity is obvious.

Digital identity is unique in a specific context. The user has a unique identifier to refer to the user in the target system. The user might have multiple attributes and roles as name-value pairs within the system and depending on those, the user has access to different resources. The aggregation of user attributes is used to create a profile of the user in the system.

Digital identity has levels of precision on how the user is identified. The older, but also widespread and standardized term, level of assurance (LOA), is nowadays encouraged to be replaced with terms Identity Assurance Level (IAL), Authenticator Assurance Level (AAL) or federation Assurance Level (FAL) [10] depending on the system.

Identification, Authentication and Authorization

Identification is an action to identify person. Identification can happen by combining one or more person's traits, person's identifying attributes and some proofs that the person possesses and are issued to the person by recognized authority. Person could, e.g., prove their name, passport number and date of birth by showing their passport to an airport employee. In digital world, identifying might happen first physically by proving identity to an official and after identification process person gets digital identifier and credentials which can be paired with the identifier. Some banks require such method. There are also other service providers which require knowing their customer. The level of proof varies and options for proving identity could be, e.g., utilising web camera on a live session, recording or sending documents or photos of documents of the person to be identified.

Authentication is an identity concept which means verifying that the person is who they claim to be on a sufficient certainty the service in question requires. This is conducted by exchanging credentials, like username and password combination, with the target system. Authentication has different levels of assurance, weak or strong. Strong assurance of authentication can utilise banking credentials or different electronic authentication applications. For example, strong authentication is used by a governmental service where person could apply for a new passport. Before applying, the person needs to authenticate with bank provided credentials to verify their identity. Thus authentication has a prerequisite that some identification or profile construction of the subject has taken place earlier. Weak authentication is used in, e.g., a web-forum application where anyone can create an account where only requirement is an email address and user provided password.

Authorization is action to grant or reject access to certain requested resource. For example, in OAuth2 framework [6], the scope is a list of resources that are requested to be released and can include username, first name or date of birth. The scope is preconfigurable in the authorization server. It is common that the scope available to different resources is different by each use case. Authorization to grant access to certain resources are typically done after authentication.

Claims and Verifiable Credentials

Claims are attribute-value pairs describing an entity on some level. Claim can be self-issued, e.g, user makes up a username or provides a drawing for a profile picture for

themselves. Claim can also be a collection of attributes. Furthermore, claim can be issued by a third party, claim can be a proof of a membership of a gym, identifier for an online video streaming service or a named local football team season ticket. These claims can be such that their issuer is not accepted by any other verifier, but by themselves. In real world this is common. For proving their identity in a bank, a person cannot just show a gym membership card, but proving ownership of the membership for the gym in question, gives access to the gym.

Verifiable Credentials are digital counterparts of real world credentials such as passport, drivers' licence or insurance card. Furthermore, they are certain type of credentials for which authenticity and integrity can be verified. The issuer of a verified credential is typically from a trusted third party and the credential digitally signed by its issuer. The signature of the credential provides the means to verify and validate the credential in question. The service provider has its own policies on which kind of credentials are valid for its services and who are the trusted third parties that can issue such credentials. Verifiable credentials are also called as assertions and the process of verification of the credential, is called as attestation [3]. These terms are also familiar from federated identity model and especially the systems utilising SAML[4] as the base technology.

For example, potential future employer might approve a certain university diploma as a verifiable credential as a proof that the applicant has graduated from the said university. The university has provided and signed the verified credential to the applicant and the service provider, the future employer, can verify the signature of the credential of the university with their public key.

Verifiable credentials are key building blocks in Self-Sovereign Identity and they will be discussed in more detail in the next chapter.

Issuer and Trusted Third Party

Issuer is the party issuing credentials and claims for the user. User passes them on to verifier, also known as relying party or service provider. Verifier is consuming the claims. Claims can be self-issued. For example, the verifier may be a web-forum which accepts pseudonyms as usernames. The verifier may have a policy that the users can provide their own username and that is sufficient information for that service.

Issuer can be a *Trusted Third Party (TTP)*, e.g, a governmental organization issuing

verifiable credentials such as digital identity cards, passports or driver's licenses. A TTP is also commonly known and also trusted by other parties in a similar context. Services may have policies such that the credentials must be issued by a certain TTP in order for accessing or continue using the service. Such services include healthcare services, banking services, rental services and many others where the users real world identity needs to be assured on high level.

Public Key Infrastructure

Public Key Infrastructure (PKI) is a fundamental concept in Internet related to validity of public keys and the identities behind them. The identity belongs to a real person or to a company. Public keys are stored in a database paired with a unique identifier. The PKI have other responsibilites, besides key registration, which include:

- Updating the public key for identity
- Querying for the public key corresponding to identity
- Verification of the public key ownership for identity
- Validating the public key
- Revoking the public the public key for identity.

The PKI is utilised in everyday Internet traffic everywhere, e.g., in e-commerce, banking, secure connections between web services and in secure email and in digital signatures. The use of certificates is a significant in PKI. Certificate contains the public key of the owner and additional information such as the purpose for the certificate, issuer, expiry date and other metadata. The certificate can be issued for a person or for company. The valid certificate is signed by a trusted authority, typically with an intermediate authority, Certificate Authority (CA). The certificate signatures form a signature chain so that also the validity of the signature of the CA's certificate can be verified from its corresponding signature. The verification chain continues all the way to root certificate at root CA, a special certificate and trusted authority, signed by itself and with ability to sign intermediate authority certificates. The certificates in this system are of type X.509 standard.

3 Self-Sovereign Identity

Self-Sovereign Identity (SSI) is an ambitious idea to shift service provider centric digital identity management to distributed model. It has potential to provide tamper-resistant, reliable, secure and privacy-protecting solution to digital transactions over different online services. SSI enables user to login to services and control as well as permit all the claims the services require. SSI also enables user to create own credentials and provide digital signatures for themselves, for other people, e.g., family members and also on behalf of a company. The SSI model opens new ways to use machine identities, e.g., in Internet of Things (IoT) related automated processes. There are proposals for required features in SSI [3] [7] [8] and some of the SSI features align [9] [11] with the European Union general data protection regulation (GDPR) [12], e.g., security, user-control, integrity, confidentiality and minimal disclosure.

Academia has become very interested in SSI, but also the commercial parties are increasingly involved with the model. There are brands with different SSI implementations, e.g., Veres One [13], Jolocom [14], Sovrin [15], Microsoft (Microsoft Entra Verified ID) [16] [17], Veramo [18] (open source, split from uPort [19]) and Serto [20] (commercial, split form uPort). European Union has its own projects regarding SSI, e.g., a blockchain initiative [21] and a digital identity card and wallet, European digital identity (eID) Wallet, for each member state [22].

The figure, 3.1, describes the top level model of all the parties involved in the processes of creating, using, revoking, storing, validating and verifying the identities in the SSI model. The arrows in the figure imply trust.

3.1 Holder

The term holder describes the user-entity better than the user, end-user, subject or owner. This is because the SSI model supports passing on or delegating credentials. Holder is normally user, but holder could also be a legal guardian for another person, for an animal, or holder could be authorized to act as behalf of another person or a company.

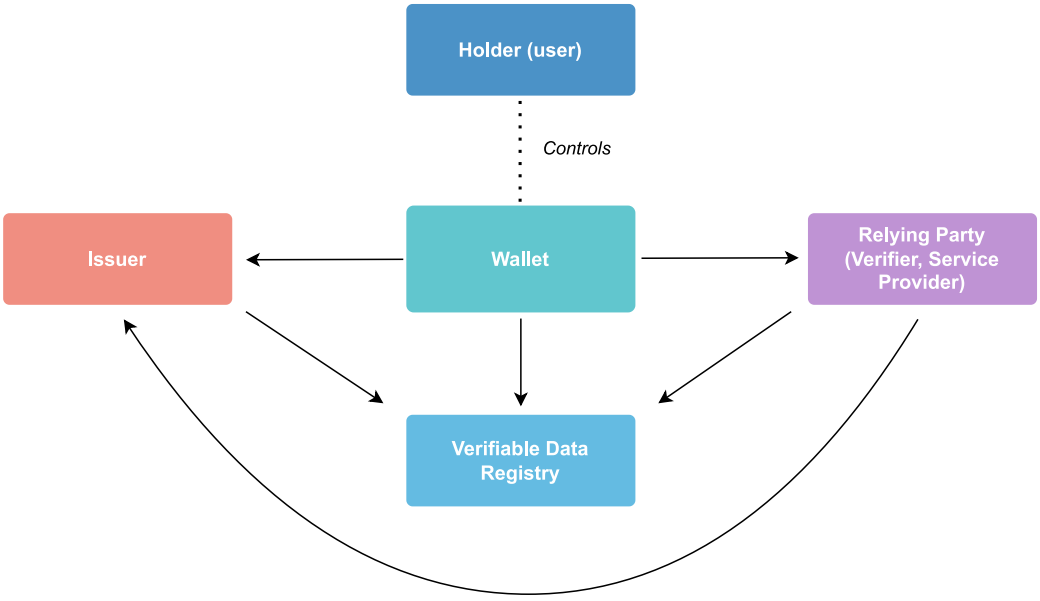

Figure 3.1: Self-Sovereign Identity parties

3.2 Verifiable Credentials

Verifiable Credentials are tamper evident, digital credentials acting in the digital world similarly as their physical counterparts act in the physical world. They prove their holder's right to something, right to entry, right to do something or right to travel.

The Verifiable Credentials data model are a World Wide Web Consortium (W3C) recommendation [23], which may be cited as W3C standards. The Verifiable Credentials are essential components in whole SSI model. This section inspects some technical aspects of a verifiable credential. The following contains descriptions of the basic concepts and properties with syntactic example of Verifiable Credentials by the W3C standard.

The verifiable Credentials implementation syntax for these examples are from JSON (JavaScript Object Notation) [24] extensions, JSON-LD (JSON Linked Data) [25] and JWT (JSON Web Token) [26] specifications. It is important to note that the verifiable credential can be in plain JSON format in addition to JSON-LD for wider support.

The 3.1 Verifiable Credential Example, is an example of verifiable credential with basic properties included.

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
```

```
    "https://www.w3.org/2018/credentials/examples/v1",
    "https://w3id.org/security/suites/ed25519-2020/v1"
  ],
  "id": "http://example.edu/credentials/1019",
  "type": [
    "VerifiableCredential",
    "UniversityDegreeCredential"
  ],
  "issuer": "https://example.edu/issuers/21",
  "issuanceDate": "2020-05-31T17:01:53Z",
  "expirationDate": "2030-05-31T17:01:53Z",
  "credentialSubject": {
    "id": "did:example:acfef236ceba5f29aec34190aff",
    "degree": {
      "type": "BachelorDegree",
      "name": "Bachelor of Science"
    }
  },
  "credentialStatus": {
    "id": "https://example.edu/status/49",
    "type": "CredentialStatusList2017"
  },
  "proof": {
    "type": "Ed25519Signature2020",
    "created": "2022-11-30T18:27:04Z",
    "verificationMethod": "https://example.edu/issuers/18#key
      -1",
    "proofPurpose": "assertionMethod",
    "proofValue": "0xCazHb..."
  }
}
```

Listing 3.1: Verifiable Credential Example

Context

The context (`@context`) property defines the vocabulary and terms for this verifiable credential, e.g, which type of credential it is and which type of attributes it might have. Context property is an ordered set, where the first item must be a Uniform Resource Identifier (URI) with the value `https://www.w3.org/2018/credentials/v1`. This value points to base context of verifiable credentials. Subsequent items in array must contain relevant information about the context and they can be either a URI or a valid JSON object.

Identifier

Identifiers, if they exist, in verifiable credentials are to be always single-valued and in form of URI. Preferably the URI contains information about the segment it is identifying. The `id` field is not mandatory. In 3.1 Verifiable Credential Example, field `id` is used in two segments. The first one is related to the type of the credential and the latter is related to the credential subject. The latter is also in form of a *Decentalized Identifier, DID*, which is covered in detail in this chapter.

There can be requirement for privacy protecting use cases, e.g., where data of person's health are handled. In these cases, the correlation based on the identifiers should be prevented. Some of the recommendations for anti-correlation include: replacing identifier by short-lived single-use bearer token, binding identifier to a single origin and implementing single use identifiers. These techniques helps to prevent any combine efforts to correlate identifiers by some parties in the system, e.g., issuers and verifiers.

Type

The `type` field is mandatory for verifiable credential and it must be a URI or the `@context` must have the mapped information about the type. The type classifies the verifiable credential to a certain type. In 3.1 Verifiable Credential Example, the `type` of credential is of *VerifiableCredential* and of *UniversityDegreeCredential*.

Credential Subject

The `credentialSubject` is an object containing one or more *claims* about the subject, e.g., the holder. The `credentialSubject` can also have multiple subjects, e.g., credential subjects can be child's parents. Credential subject may have an id, or it can be omitted, e.g., for privacy preserving purposes.

Issuer

The `issuer` in Verifiable Credential is a mandatory field. The issuer can be a URI or a valid object with `id`. Object allows extending the Issuer with other properties, e.g., name of the university. The specification recommends that when URI is used, the URI should contain information about the entity in proper machine readable format, such as JSON. This recommendation holds on all the cases where URI is an option for value. Other possible values for issuer include DID or a URI to a cryptographic key, JSON Web Key (JWK).

Issuance Date

The `issuanceDate` is a mandatory field in Verifiable Credential. And it must be in form of Coordinated Universal Time (UTC), XMLscheme11-2 [27]. In 3.1 Verifiable Credential Example, `2020-05-31T17:01:53Z`, consists of:

```
Year '-' Month '-' Day 'T' Hour ':'  Minute ':'  Second 'Z'
```

The character 'T' separates the date types from time types. The character 'Z' is an offset of '00:00' UTC. This field is subject to change in favor of 'issued' property, which would also include 'validFrom'.

Proof

There needs to be at least one `proof` mechanism available in Verifiable Credential. The `proof` field contains mechanism which can be external, wrapped in JWT or embedded, as is the case in, 3.1 Verifiable Credential Example. Proofs are digital signatures and there are no standards defined for VC on how or which mechanism is to be used. The specification itself has listed three different mechanisms, but not all of the three are standardised at

the time of writing: JWTs [26] secured using JSON Web Signatures (JWS) [28], Data Integrity Proofs [29] and Camenisch-Lysyanskaya Zero-Knowledge Proofs [30]. If JWS, `jws` is present, the Proof propertu can be omitted. The JWS is enough to prove the integrity if the Verifiable Credential and Verifiable Presentation. Then the issuer should be issuer of the Verifiable Credential or issuer of the Verifiable Presentation. The signature mechanisms need extra care, when the signer's privacy is a concern or when there is a requirement for anti-correlation. Note that in 3.1 Verifiable Credential Example, the proofs' signature is of type, `Ed25519Signature2020`, Edwards Curve Digital Signature Algorithm (EdDSA) and Curve25519 (ed25519) [31]. The signature type is accepted by the U.S. National Institute of Standards (NIST) in the latest FIPS 186-5 and in conformance with the Verifiable Credential Data Integrity 1.0 document [29]. The latter is a draft document describing the use of digital signature mechanisms with Verifiable Credentials and other similar documents. However, the `Ed25519Signature2020` must not be used if anti-correlation is a requirement, since the cryptography suite does not support anti-correlation or selective disclosure.

Expiration

The `expiration`, is not mandatory. Currently the expiration goes by name, `expirationDate`, but is bound to be replaced by, `validUntil`, property.

Status

The `credentialStatus` is a optional property in verifiable credential. However, it can be used to express the current state of the credential, e.g., if the credential is revoked or suspended. At the time of writing, the specification did not list more values for the status. If `credentialStatus` is present, it must have an `id` and a `type` properties. The id property must be an URI and the type, would contain machine readable information, such as JSON, about the current status of the credential.

3.3 Verifiable Presentations

Verifiable Presentation Verifiable credentials (VCs) are issued to a subject and the Verifiable Presentation is a document signed by the subject of one or more verifiable cre-

dentials. This document is presented on request to a verifier, e.g., the service provider to consume. The VCs can be from multiple different issuers. Verifier's policies dictate the valid issuer for each case. The verifiable presentation typically includes: `id`, `type`, `verifiableCredential`, `holder` and `proof`.

3.4 Decentralized Identifiers

The Decentalized Identifiers (DIDs) [32] are new type of unique identifiers owned by their creators, individuals and organizations. There is no central party in control of the DIDs like there are for Uniform Resource Locators (URLs), phone numbers or social security numbers. Every DID ownership can be cryptographically verified by digital signatures and there can be arbritrary quantity of DIDs generated. The DIDs can be used for proving ownership of data.

Every valid DID resolves to DID Document with matching identifier. The creation of a DID starts by having access to an asymmetric key pair. It could be already generated at some or it could be generated when needed. Then the DID document can be created. The DID document is a set of data including the owner identifier of the document and set of public keys and some other metadata, e.g., about web domain ownership or web service endpoints. The public keys of the DID document can be used for authentication [33] and prove the ownership of the DID. The private key must be possessed by the controller of the DID and the DID document can be published to, according to selected DID method, to a location where it can be accessed for the document consumers. The location for the DID can be, e.g, a blockchain or a web server or any other registry. The DID must be resolvable, i.e., when the DID is presented for a service, the service verifies the DID

Figure 3.2: Decentralized Identifier (DID) example

The DID is comprised of the `scheme`, the `DID method` and the `identifier`.

The `scheme` signifies of the protocol which is used to deal with the data regarding the DIDs. This is similar to other syntaxes in protocols, e.g., File Transfer Protocol (*ftp:*) or the Hyper Text Tranfer Protocol (*http:*).

The `DID method` describes the Decentralized Identifier method which to use to interact with the DID documents. There are multiple different DID methods with various mechanisms and rules to store, retrieve and verify the contents of the DID documents. Many of the DID methods utilise some blockchain distributed ledger technology as their Verifiable Data Registry. The distributed ledger stores all the transactions of the DIDs in to the ledger which everyone has read access to verify the transactions. The transactions include creation, reading, updating, and deactivation (CRUD) of DID documents.

There are already 170 registered DID methods on DID Specification Registries [34] and the real number is assuming since these are only the ones which have registered their DID method. Anyone can create a DID method. The `identifier` must be unique and must not change. It can contain a query, e.g.,
`did:example:acfef236ceba5f29aec34190aff?versionId=2` or inclusively a fragment `did:example:acfef236ceba5f29aec34190aff#key-1`. The fragment is commonly used when DID is set point a certain public key for verification. The DID document can act as an endpoint for verifying the ownership of keys by verifying the signature of a message with key locating in the DID document.

Some examples DID methods include:

- `did:btcr`, The main Bitcoin blockchain, all the transactions require Bitcoin currency (BTC) to interact with it's blockchain. This is a proof-of-work type of blockchain;
- `did:ethr`, The Ethrereum blockchain. The interactions with it's blockchain require consuming *gas*, which can be obtained with Ethereum currency (ETH). Ethereum blockchain transformed from proof-of-work to a proof-of-stake type of blockchain in September 2022;
- `did:web`, The DID web method, which anyone with a control of web domain name and a hosting service can utilise. The DID documents need to be stored on server under `well-known` endpoint in a JSON format. The in a manner which can be freely chosen by the host of the DID document. Few examples of the DIDs their DID Document locations are:
  - a general DID for the example.com domain, `did:web:example.com` ->

`https://example.com/.well-known/did.json`;

- a DID for a specific person `did:web:example.com:user:alice` -> `https://example.com/user/alice/did.json`;
- a DID with specific port number. The colon character ':' before the specifying the port number must be urlencoded in the DID to avoid collision with the DID syntax, `did:web:example.com%3A3001:user:bob` -> `https://example.com:3001/user/bob/did.json`.

There are issues with the DIDs of the `did:web` type, e.g., web domains change ownership, web domains cease to exist, web server control issues and also the improper DNS configurations might compromise the DID documents.

- `did:ethr:goerli`, This the Ethereum blockchain's test environment. The developers test their programs and logic in test environment before releasing the solutions in, e.g., `did:ethr`. This DID method requires also currency (goerli testnet goETH) to use, but unlike BTC or ETH, the goETH can be obtained freely. The developers of Ethereum blockchain are about to deprecate goerli in favor of Sepolia;
- `did:key`, This is a static method for storing DID documents [35]. With this method the DID document is included in the DID method specific identifier and it is not stored in any blockchain nor web server. For this reason the the DID lacks common DID methods of updating or deactivating the DID, i.e., any update to DID document would result new DID method specific identifier, which is completely new unique DID. Because this DID method does not support key rotation, long term support is discouraged.

The 3.2 DID Document Example is a valid, resolvable DID Document with DID `did:ion:EiAtFwCx-lUsmhpDzV2e_X3iZUyqMcl-R8tJC5JyNxHpgw`.

```
{
  "id": "did:ion:EiAtFwCx-lUsmhpDzV2e_X3iZUyqMcl-
     R8tJC5JyNxHpgw",
  "@context": ["https://www.w3.org/ns/did/v1",
    {
      "@base": "did:ion:EiAtFwCx-lUsmhpDzV2e_X3iZUyqMcl-
         R8tJC5JyNxHpgw"
    }
```

```
  ],
  "service": [{
      "id": "#domain-1",
      "type": "LinkedDomains",
      "serviceEndpoint": "https://foo.example.com"
    }
  ],
  "verificationMethod": [{
      "id": "#key-1",
      "controller": "did:ion:EiAtFwCx-lUsmhpDzV2e_X3iZUyqMcl-
         R8tJC5JyNxHpgw",
      "type": "EcdsaSecp256k1VerificationKey2019",
      "publicKeyJwk": {
        "kty": "EC",
        "crv": "secp256k1",
        "x": "H6dqfRcNoUNvJyJcSMWNf3UEcvdnSIBB4hy0YGxxj2M",
        "y": "kFz6sh2kXeWgHxCgcCGp1YcmGgYqYSX4P6BMkTMBzJg"
      }
    }
  ],
  "authentication": ["#key-1"]
}
```

Listing 3.2: DID Document Example

The 3.2 DID Document Example has one public key in the `verificationMethod` with id, `#key-1`. The DID has also one `service` with id, `#domain-1`, which has one `serviceEndpoint` for URL `https://foo.example.com`. The `authentication` refers to `#key-1`.

The holder of the corresponding private key could now sign messages, e.g., JSON Web Tokens (JWT). By and setting the header field, `kid` (key id) of JWT as `did:ion:EiAtFwCx-lUsmhpDzV2e_X3iZUyqMcl-R8tJC5JyNxHpgw#key-1`, the verifier could resolve the DID and dereference the corresponding public key from the DID document and verify the signature.

3.5 Digital Wallets

Digital Wallets are software agents, e.g., native mobile or cloud applications for handling the user's Verifiable Credentials management. The management include, e.g., storing the credentials, generating verifiable presentations on request and approving and revoking the credential access for relying parties.

3.6 Verifiable Data Registry

The Verifiable Data Registry (VDR) is referred to as a system where the DID documents are persisted. The VDR and the APIs to execute operations for the DID documents, e.g., creation, reading, updating, and deactivation (CRUD), varies by the solution. There is no strict definition of the type of what kind of architecture the VDR has to be. The VDR can be, e.g., a blockchain system, a database, peer to peer solution, a web server or other type of distributed file system [36]. Likely it will be a combination of the mentioned, since vast amount of use cases and trust issues. There can be government hosted filesystem for some government created DID documents where persistence and availability is a requirement. And on the other hand there can be more leniently created short-lived DID documents hosted on temporary web servers.

Blockchain

Blockchain is a distributed technology which consists of network of nodes. The nodes in the network act as a distributed database consistently replicating the data with each other using Distributed Ledger Technology (DLT). The blockchain technology was made well known to general public by the use of cryptocurrency such as *Bitcoin* or *Ethereum*. The blockchain itself is a linked list of data objects, called blocks. The blocks include information about transactions, e.g., how much currency was moved, from which address, the DID document and the key rotation update to the DID document. Each block has a timestamp and it cryptographically signed by the owner, i.e., party responsible of the transaction. Every transaction, i.e., new record to blockchain is appended to the blockchain. The transaction is verified by the consensus mechanism depending of the type of the blockchain.

Apart from their obvious use cases of handle digital payment processing, the blockchains can be used to store any type of data and they have also other features. These features

include:

- the immutability, every transaction, i.e., what is written in the blockchain, stays there unchanged;
- non repudiation, every transaction must be digitally signed;
- decentralized, there is no central authority in control of the blockchain.

The blockchains comes in different types what comes to their consensus mechanisms and their access controls. Consensus mechanisms include, e.g., *proof-of-work* and *proof-of-stake*.

In the *proof-of-work* blockchain, the participant nodes has to solve a cryptographic puzzle, e.g., the hash value of the next transaction. The computation gets more difficult as more blocks are added to blockhain and more nodes join the network to solve the puzzle. The *Bitcoin* runs on *proof-of-work* blockchain.

In the *proof-of-stake* blockchain, the consensus of the blockchain is reached by selected trusted validators in the network. Blockchains are a promising candidates for Verifiable Data Registries (VDRs) in the Self-Sovereign Identity model and are already well utilised as VDR for the DID documents and a very. There are different DID methods using blockchains, e.g., `did:btcr`, `did:ethr`, `did:sovr` and `did:ion`.

Due to the immutability nature of the blockchains, it must be made sure that no private information, e.g., personally identifiable information, should be written in blockchain. The transactions in blockchain can be traced and deduction can also be made about the party who has made the transactions.

4 Feasibility study

This chapter presents methods of extending an open source software OpenID Provider (OP) to be able to utilize Verifiable Credentials and Verifiable Presentations in authentication and authorization. The OP acts as a verifier in this context. The goal of the study is to assess the feasibility of utilizing web application Wallet, issuing Verifiable Credentials, storing them in to the Wallet, wrapping the Verifiable Credentials in Verifiable Presentation and present it back to OpenID Provider, which initiates a Single-Sign On session with Relying Party, the service provider.

This study will assess two different methods to utilize Verifiable Credentials and Presentation with OpenID Connect. In the first one, 4.1 Federated Verifiable Presentation Flow, the OpenID Provider will verify the Verifiable Credentials and the Verifiable Presenta-

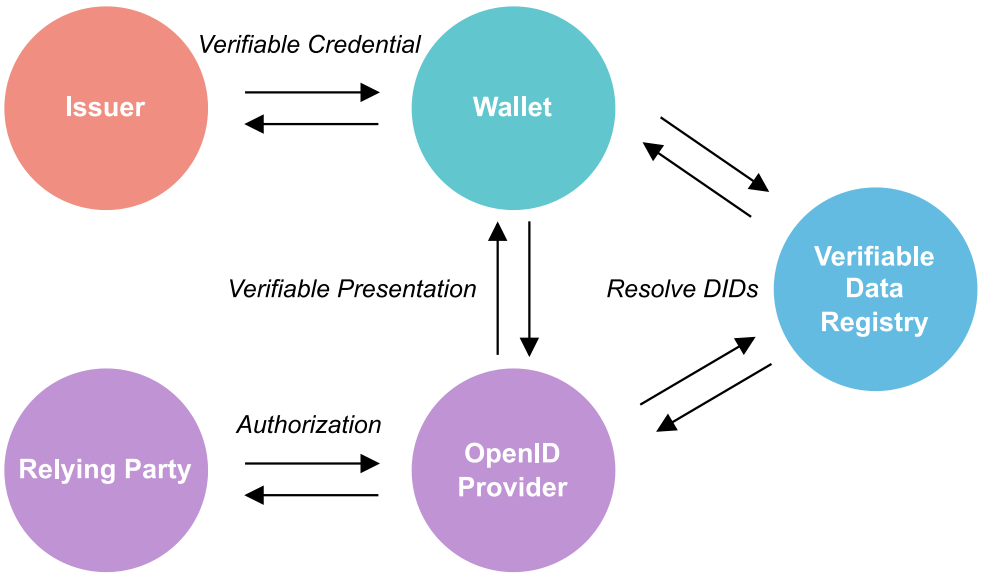

Figure 4.1: Federated Verifiable Presentation Flow

tion on behalf of the Relying Party and approve or disapprove the flow accordingly. The OpenID Provider will act as a trusted third party and identity broker, which will validate and verify Verifiable Presentations and resolve Decentralized Identifiers (DIDs) coming from the Wallet. The Wallet hosts Verifiable Credentials issued by the Issuer. Wallet approves or disapproves requests from the OpenID Provider and the Wallet will validate, verify payloads and resolve DIDs coming from OpenID Provider. In this flow the Relying Party trusts OpenID Provider and only verifies the tokens which it receives from the

OpenID Provider.

The second one, 4.2 Verifiable Presentation Flow, leaves the OpenID Provider out and the Relying Party communicates directly with the Wallet. The Relying Party will now validate and verify Verifiable Presentations and resolve DIDs coming from the Wallet. And Wallet validates and verifies the request payloads and DIDs coming from Relying Party.

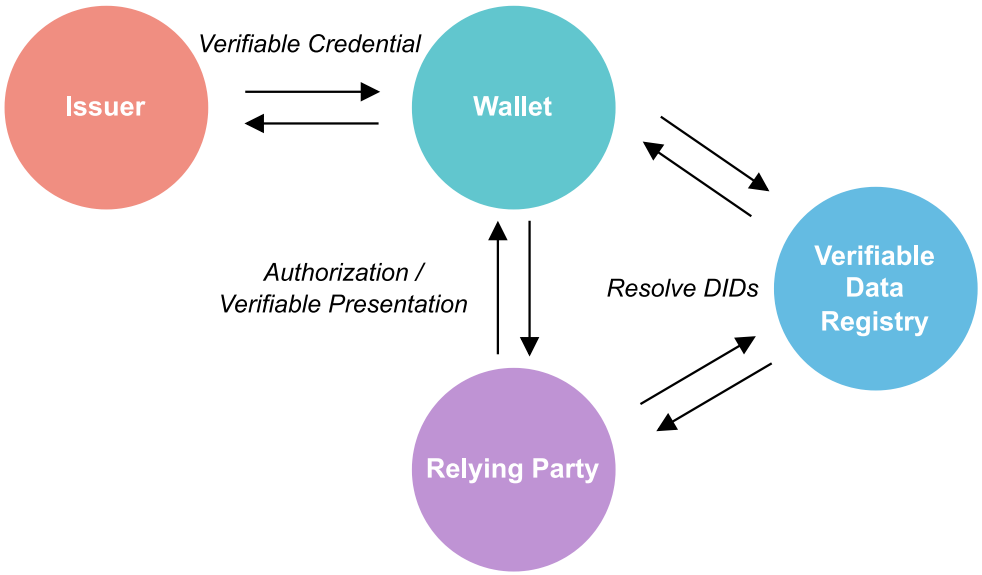

Figure 4.2: Verifiable Presentation Flow

The flows will follow OpenID Connect protocol customized to issue Verifiable Credentials, include them in Verifiable Presentation and use the it for login to service. The study follows the specifications of the OpenID standards track of OpenID for Verifiable Credential Issuance [37] and OpenID for Verifiable Presentations [38]. The Wallet application follows the standards track of Self-Issued OpenID Provider [39].

Following sections will present the software and the components used in the study.

4.1 The Software and Components

Following software components of this study are custom made for this study: the `Issuer`, the `Relying Party` and the `Wallet`. The technology behind all of the components is NodeJS, i.e., JavaScript. The purpose of these components is not to provide a full working production grade applications, rather than highlight their purpose in this study as working components. They work as intended in the scope of study, but they lack many features which would be required to use in any kind of production environment.

The OpenID Provider is a fork from Panva Openid Provider and is modified for the purpose of this study. Any of the introduced modifications have not been audited and it is not advisable to use that fork in any production environment.

The whole implementation of the Issuer, OpenID Provider, the Wallet and the Relying Party, which are produced or modified for the feasibility study are referred to as OpenID4VC [40] from now on.

JSON Web Tokens

The JSON Web Tokens (JWTs) are constructed from three parts: `header`, `body` and `signature`.

```
{
  "typ":"JWT",
  "alg": "ES256K",
  "kid": "did:ion:EiAtFwCx-lUsmhpDzV2e_X3iZUyqMcl-
     R8tJC5JyNxHpgw#key-1"
}
```

Listing 4.1: JWT Header Example

```
{
  "sub":"1523",
  "name":"Alice Bobson"
}
```

Listing 4.2: JWT Body Example

The signature is formed, e.g., using `ES256K` algorithm and with the contents of 4.1 JWT Header Example and 4.2 JWT Body Example. The `header`, `body` are `Base64Encoded` and signed with `privateKey` of the asymmetric key pair. The signature is an array of bytes, which should be `Base64Encoded` to be readable. The signature can be verified with the corresponding `publicKey`. The 4.3 JWT Creation Example describes this process and 4.4 JWT Example shows the result.

```
signature = ES256K(
    BASE64ENCODE( header ) + "." +
    BASE64ENCODE( body ), privateKey
```

```
    )
JWT = BASE64ENCODE( header ) + "." BASE64ENCODE( body ) "." +
    BASE64ENCODE( signature )
```

Listing 4.3: JWT Creation Example

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJFUzI1NksiLCJraWQiOiJkaWQ6aW9
   uOkVpQXRGd0N4LWxVc21ocER6VjJlX1gzaVpVeXFNY2wtUjh0SkM1
   Snl0eEhwZ3cja2V5LTEifQ.eyJzdWIiOiIxNTIzIiwibmFtZSI6
   IkFsaWNlIEJvYnNvbiJ9.f3XE0gn6I9IbxuuJcSYaQqVGfb9
   psLuySfaIXfndNT8emRL_oH2c2on6P0DzqbYxvcbc3BvM3wxdcmJcL0fRYQ
```

Listing 4.4: JWT Example

The The OpenID Connect (OIDC) servers hosts traditionally the public keys in standard `jwks_uri` endpoint [5], where the consumers of the JWTs, e.g., Relying Parties can acquire them and verify the signature of the token. In this study, the OpenID Provider has one public key in such endpoint, which is not used with Decentralized Identifiers (DIDs). With DIDs, the `jwks_uri` endpoint is not used. The DID must be resolved to a DID Document which contains the public key of the signed payload, i.e., token. The OpenID Provider is also using a DID, but in the federated use case, the Relying Party receives standard OIDC `id_token` and for that reason the `jwks_uri` endpoint is available. The DID `did:ion:EiAtFwCx-lUsmhpDzV2e_X3iZUyqMcl-R8tJC5JyNxHpgw` can be resolved as DID document, which is the same as 3.2 DID Document Example and the `key1` of the document contains the JSON Web Key (JWK), `publicKeyJwk` which can be used to verify signature of the 4.4 JWT Example.

Resolution of Decentralized Identifiers

Resolution of Decentralized Identifiers (DIDs) means producing the DID Document and metadata referred by the DID with the DID Method specific rules. For this study, the DID Documents are the main focus of this action. The DID Documents hosts the public keys to verify different digital signatures throughout the study. The DID Methods have different ways of resolving, reading, updating and deleting the DID Documents. This study uses the `did:ion` DID Method. The libraries [41] used in this study can not resolve any other DIDs than the `did:ion`. In practice, the resolvers should be able to resolve multiple different and popular DIDs. Every component that resolves DIDs, should implement their own resolvers, rather than rely on third party resolvers, because of security risks.

Issuer

The OpenID4VC includes an implementation of a self-made web application for the Issuer of Verifiable Credentials (VCs). the Issuer is cabable of issuing `IDCardCredential` containing `given_name`, `family_name` and `date_of_birth` attributes. The Issuer could be any governmental organization authorized to issue `IDCardCredential`. The Issuer also requires the Wallet DID because that will be the subject of the credential issuance. Note that, if the Wallet DID given for Issuer does not match the DID of the implemented Wallet, the Wallet rejects the VC. Other attributes, `given_name`, `family_name` and `date_of_birth`, can have any values.

The implemented Issuer does not have any authentication. Normally there would have to be strong authentication mechanism in place since the Issuer is issuing an `IDCardCredential`. The authentication would be done using, e.g., 4.3 Authorization Code Grant Flow, but between the Wallet and the Issuer. The last step of the 4.3 Authorization Code Grant Flow the Wallet would send the request to Issuers `/credential` endpoint instead of the `/userinfo` endpoint for the credentials issuance [37].

The DID for Issuer is `did:ion:EiBAtbiEe2qtLsa5a9_fgPQDUAtxBKXLvpI6Lvpkdrcobg`.

Relying Party

The OpenID4VC includes an implementation of self-made web application for Relying Party (RP). The RP is a web page, which requires user to login. The RP could be any web application requiring a verified first name, last name and date of birth from the user. On successful login, the RP shows the user information. The RP has an OpenID Connect Client configured in OpenID Provider and the RP is able to utilise the 4.3 Authorization Code Grant Flow with OpenID Provider.

The user information can be received either from OpenID Provider which receives it from Wallet, utilising 4.3 Authorization Code Grant Flow or 4.4 Federated Verifiable Presentation Flow, or directly through Wallet, utilising 4.5 Verifiable Presentation Flow. The two of the latter flows are designed and implemented as part of the study.

The Relying Party did not require any changes when utilising 4.4 Federated Verifiable Presentation Flow compared to 4.3 Authorization Code Grant Flow. This was an important observation regarding bringing the 4.4 Federated Verifiable Presentation Flow in to

use. All the required changes were made to the OpenID Provider. Changes includes all the required requests, payload constructions, validations and response handling, which are discussed in detail in 4.4 Federated Verifiable Presentation Flow.

When utilising the 4.5 Verifiable Presentation Flow from RP, changes were the same that were required in the OpenID Provider with 4.4 Federated Verifiable Presentation Flow. When using direct login with wallet, Relying Party has a policy check for trusted Verifiable Credential Issuers.

The DID for the Relying Party is
`did:ion:EiBQsxvT1tz0Cz7KEfFuJhJt_134d_suJlwZ3S_bXVnoBA`.

Wallet

The OpenID4VC includes an implementation of self-made web application for the Wallet application. The application is a proof-of-concept type of wallet and lacks many of the security features, e.g., access management to wallet. This would be typically done by username and password combination, biometric authentication or with other types of authentication such as passkeys or FIDO2. The wallet is done as a Self Issued OpenID Provider (SIOPv2) [39] for the relevant parts. This wallet application responds only with verifiable presentations, i.e., normal id token responses are out of the scope of this study. There is no pre-registered OpenID Clients in the Wallet, the OpenID Connections are dynamic and the Wallet follows the Decentralized Identifiers of the SIOP specification [39]. That is, where the Relying Parties, or OpenID Provider, are providing their DID as the `client_id`. The Wallet has a JSON file database, which is more than enough for the feasibility study, but would have to be replaced with properly managed database.

The Wallet is able to collect and store Verifiable Credentials from the Issuer, inspect the credential request from the OpenID Provider or Relying Party, and it is able to create a signed Verifiable Presentation to present the credentials.

The DID for the Wallet is
`did:ion:EiAZF8EUsRKvkuqQeAvtha8WnxFa2VlK3M7XwJ1EsOfEvA`.

OpenID Provider

The OpenID Provider used for this study is modified from the Panva's oidc-provider [42]. Panva is the Github nickname of the project's author. The oidc-provider is an open

source project and is forked and modified for the feasibility study. The oidc-provider is highly customizable NodeJS project that includes multiple features compliant with OpenID Connect specification. The complete source code is available and it is a good candidate for the baseline of this study. It will then be extended to support Verifiable Credentiasl (VCs), Verifiable Presentation (VPs) and Decentralized Identifiers (DIDs). The code base is to be extended to match the current specifications for OpenID Connect for VPs and VCs. For clarity, the modified project is referred to as OpenID Provider. The forked repository and the extended project is found in [40].

The OpenID Provider uses modified endpoints and OpenID Client to enable the DID utilisation for federated authentication with the Wallet. The OpenID Provider is able to craft a request object requesting Verifiable Presentation when initialising the federated authentication. The OpenID Provider is able to sign the JSON Web Token payloads and resolve the DIDs and verify the received payloads with public keys found in resolved DID Documents. The OpenID Provider is also able to initiate Single Sign-On (SSO) session for the Relying Party and it has simple trust control mechanism for trusted Verifiable Credential Issuers.

The DID for the OpenID Provider is
`did:ion:EiBr3cl0yOq4TDkQ-AioibD8NF2Miml3BQ-40smk5Viu0Q`.

Verifiable Data Registry

The OpenID4VC utlises Identity Overlay Network (ION) [43] and its programming libraries [41] to interact with the Verifiable Data Registry.

The DID method, `did:ion`, is a middleware DID and it has a database running on top of Bitcoin blockchain. The DID operations are done in batches and written as single transaction to blockchain. The DID Tools are open source, do not require cryptocurrency to use and it has supporting programming libraries to handle the database and blockchain interactions [41].

4.2 Authorization Server Metadata

Authorization Server Metadata is a term that describes an URI, which is an HTTP endpoint that returns information about the authorization server in JSON format. The authorization server metadata contains information about URIs which are to be used when

interacting with the Authorization Server, e.g., at given phase of authorization flow. This endpoint is typically referred to as the `well-known endpoint`. The metadata can also contain information about different supported scopes, grants and response types. One important type of metadata is information about the public keys of the server. Typically this information is referred as the JWKS (Json Web Key Set) endpoint and it contains, e.g., the key id, cryptographic algorithm family (RSA, EC) and purpose for its use (sign or encrypt). These keys are used to verify the digital signatures the server provides with the messages, typically JSON Web Tokens (JWTs). This way any relying party can verify that the message is from the OpenID Provider by verifying the signature of the message with the public key provided in the JWKS endpoint.

Authorization server can be the Wallet application and the OpenID Provider depending on the context. This chapter introduces a method where the context changes: The OpenID Provider becomes a Verifier (Relying Party or Service Provider) and Wallet is the OpenID Provider. But when the authentication and authorization are completed for the Wallet, the OpenID Provider is the again the OpenID Provider for the original Relying Party.

The provided OpenID Provider in this study follows OAuth2 server metadata specification [44]. The provided Wallet application has simplified metadata and also contains different fields related to verifiable presentations as per specification [38].

4.3 Authorization Code Grant Flow

The standard OpenID Connect and OAuth2 Authorization Code Grant Flow is used in this section to present common method to authenticate user in a federated environment. The authorization flow is useful for introducing and describing the to sequences and terms that are used throughout the work.

```
https://openid-provider.com:3000/authorize?client_id=
   clientabc123&redirect_uri=https://relying-party.com:3001/
   redirect&scope=openid profile&response_type=code&state=efg9
   87&nonce=hij456
```

Listing 4.5: Authorization Request Example

Authorization Request (Step 1)

User is required to login to a web service, and clicks login on the Relying Party (RP)

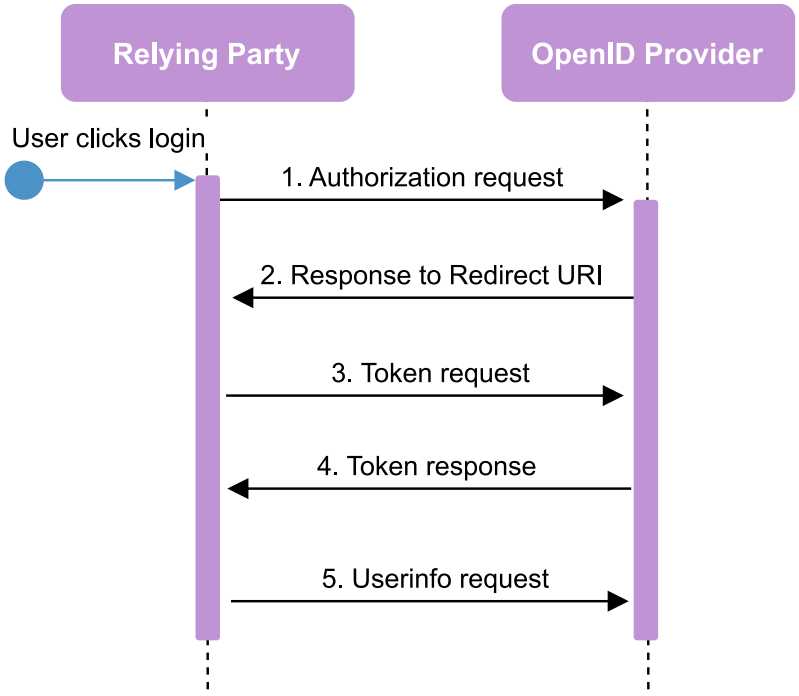

Figure 4.3: Authorization Code Grant Flow

which acts as a service provider. Web service initiates authorization, by constructing `authorization request` and sending it to an OpenID Provider (OP), which acts also as an identity provider.

The authorization request contains some parameters for the OP to verify.

- The mandatory `client_id` is a unique identifier referring to an OpenID Connect Client, a software agent used between the RP and the OP during authentication and authorization.
- The mandatory `redirect_uri` parameter is a pre-configured valid URI where the authorization response is sent with the authorization code.
- The mandatory `scope` parameter must include value *openid*, when OpenID Connect used. The `scope` contains one or more claims, user attributes, about the user. The `scope` *profile* could include, e.g., given name and family name and `scope` *email* could include on or more email addresses. The `scope` can not contain any value that is not pre-configured in OpenID Connect client.
- The mandatory `response_type` describes to server what kind of authorization response is requested.

- The recommended `state` is for security purposes to counter Cross Site Requst Forgery (CSRF) attacks. The state sent in the authorization request must match with state in the response.
- In this request, the `nonce` parameter is optional. if present, the value must match with the `nonce` value in `id_token` received by the Relying Party after successful code exchange in token response.

User needs to authenticate themself typically by providing username and password.

Authorization Response to Redirect URI (Step 2)

The user needs to authenticate themself in OP, typically with username and password. After successful authentication, user is redirected back with authorization response to RP's `redirect_uri` with the `authorization_code`.

Token Request (Step 3)

The RP makes a request to the OP's token endpoint, to exchange the `authorization_code` for `access_token`. In this step, the RP must provide also the `redirect_uri` and `client_secret` which is exchanged out-of-band between the OP and RP. In other words OP receives a request with one-time-use code, a secret value which delivered only to RP and the configured `redirect_uri` for RP. These values have been thought to be enough for the OP to trust the RP, although nowadays it is very common to use PKCE extension with the Authorization Code Grant Flow.

Token Response (Step 4)

With correct `authorization_code` and other required parameters present, the flow returns to RP with `access_token`. An `id_token` will also be present in the response, if the value *openid* was included in the `scope` of the authorization request.

Userinfo Request (Step 5)

The RP can now make requests to OP's `userinfo` endpoint with the access token. The OP returns user information from its user database within the requested `scope`. Depending on the `scope` and implementation, the user information could also be already in the `id token` received in token response. Now the RP, the service provider has confirmation that the user has provided correct login details and the RP can access to user data according

to scope requested.

4.4 Federated Verifiable Presentation Flow

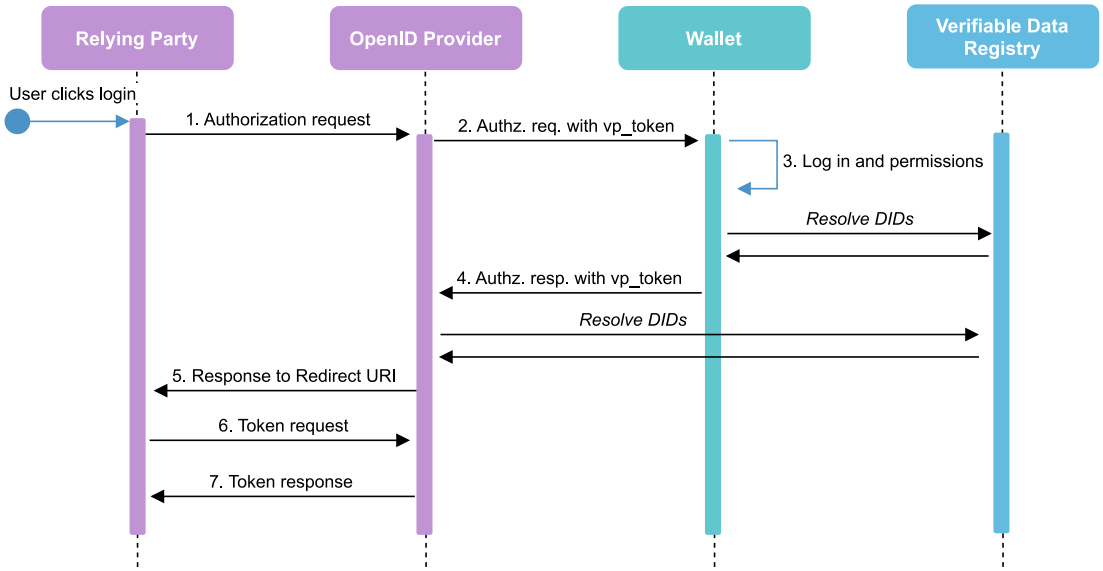

Figure 4.4: Federated Verifiable Presentation Flow

The 4.4 Federated Verifiable Presentation Flow is not part of OAuth2 or OpenID Connect, rather it is a result of combining the specifications of those two and also the draft of OpenID for Verifiable Presentations [38]. There are two possible ways to implement the flow and this section will describe the federated version of the flow. The implementation of Federated Verifiable Presentation Flow does not support dynamically acquiring the credential, if one is missing from the Wallet. The Verifiable Credential must be imported manually to the Wallet prior for successful authentication. The 4.4 Federated Verifiable Presentation Flow will be federated type [45] of implementation where the OP has delegated the identity provisioning to a trusted third party, i.e., trust anchor. In this flow, OpenID Provider (OP) is acting as a relay. The OP does not have the user data in its database and does not store it there when the wallet responds back with the verifiable presentation. Federation is widely in use with OIDC and OAuth2 systems. This flow could be seen as one phase or step for utilising Verifiable Credentials and Verifiable Presentations with OpenID Connect in registration and login purposes.

Assuming the Relying Party (RP) is already using the OP for Single Sign-On, The Federated Verifiable Presentation Flow does not require the RP to do much changes in its code base.

Instead of proof field in the `verifiable_credentials` and `verifiable_presentations` this study uses both of them as JSON Web Tokens, according to the W3C draft of Securing Verifiable Credentials using JSON Web Tokens [46]. The integrity of the credentials and presentations are proved by verifying the JWT signature.

Authorization Request (Step 1)

The RP constructs the authorization request and sends it to OP. Instead with normal username and password login, the User chooses to authenticate with Verifiable Credentials stored in the credential Wallet. There are two options to redirect the user to use the wallet: Present a QR code which user could scan with their native mobile application, or provide a direct link, e.g., to some web wallet provider. The OpenID4VC uses the direct link. The provided wallet is a proof-of-concept type of web wallet and logging in to the wallet is completely bypassed in this work. Normally, the authentication would happen, e.g., with username and password, with fingerprint or face recognition.

Federated Authorization Request (Step 2)

The begins its federated [45] authentication with another OpenID Connect client in communication between OP and the Wallet. The new OIDC Client constructs a new authorization request and sends it to the Wallet. The federation means that some or all of identity attribute provisioning is delegated to another trusted OP. Another example of such use case is that one OP provides profile attributes, e.g., given name, family name and email. The OP needs another certified identity provider for verifying the real identity of the user and uses bank provided federated login for enriching the profile with social security number provided by the bank.

In 4.4 Federated Verifiable Presentation Flow, the original OP does not provide any user attributes at all and the OP does not store the user profile permanently. The OP delegates the user attribute provisioning to the Wallet and the Wallet acts also as a trusted OP.

The original Panva's oidc-provider [42] supports federation, but it is hardcoded to support only Google login. The modified OpenID Provder [40] supports the provided Wallet application as well. Currently the Wallet federation is hardcoded in to the OpenID Provder, but with some programming effort it could be modified to a dynamic solution to support multiple different federated providers.

The OpenID Connect Client application within the OP is initiated with the authorization url of the Wallet. The `nonce`, `state` and `request`-object are created and the OpenID

Connect Client application can utilize them as parameters for the authorization request.

The Federated Authorization Request which is sent from OP to wallet is described in 4.6 Federated Authorization Request Example and it differs from the 4.5 Authorization Request Example. Instead of chaining request parameters, like in 4.5 Authorization Request Example, the contents of the request is passed in a single `request` object parameter. The value of the `request` parameter is an `id_token` signed by the OP. Notable difference is that when the Wallet receives the request, it can confirm from the `request` object signature that the request is from OP which is not case with 4.5 Authorization Request Example. The OpenID Connect (OIDC) Clients authorizing againts the Wallet are utilising the DIDs as the Client Identifier scheme [39], e.g., there are no OIDC Clients with `client_secret` configured in the Wallet, like in Authorization Code Grant Flow.

```
https://siop-wallet.com:3002/authorize?request="..."
```

Listing 4.6: Federated Authorization Request Example

The `header` and the `body` of the request object are shown in the 4.7 Request Object Example. The contents of the body in `request` object is similar to the 4.5 Authorization Request Example, but the `client_id` is a DID, which the `client_id_scheme` value indicates.

```
// header
{
  "typ": "oauth-authz-req+jwt",
  "alg": "ES256K",
  "kid": "did:ion:EiBr3cl0yOq4TDkQ-AioibD8NF2Miml3BQ-40
     smk5Viu0Q#key-1"
}
// body
{
  "client_id": "did:ion:EiBr3cl0yOq4TDkQ-AioibD8NF2Miml3BQ-40
     smk5Viu0Q",
  "client_id_scheme": "did",
  "response_type": "id_token vp_token",
  "redirect_uri": "https://openid-provider.com:3000/
     interaction/callback/wallet",
  "nonce": "acfdef3D2",
  "scope": "openid",
```

```
  "state" :"YUe87dSq",
  "presentation_definition": "..."
}
```

Listing 4.7: Request Object Example

The `response_type` now includes `vp_token` along with `id_token`. It tells to the Wallet that a Verifiable Presentation Token should be included in the response along with the `id_token`. The `nonce` is mandatory, random and has to be different on each time when new `authorization_request` is made.

The `presentation_definition` is also a new field and it is expanded in 4.8 Presentation Definition.

```
{
  "id": "vp token example",
  "input_descriptors": [{
    "id": "id-card-credential-UUID",
    "format": {
      "jwt_vp": {
        "alg": [
          "ES256K"
        ]
      }
    },
    "constraints": {
      "fields": [{
        "path": ["$.type"],
        "filter": {
          "type": "string",
          "pattern": "IDCardCredential"
        }
      }]
    }
  }]
}
```

Listing 4.8: Presentation Definition

The 4.8 Presentation Definition describes the Verifiable Presentation and the Verifiable Credential within which the OP is expecting from the Wallet. There a `format` definition `jwt_vp`, which tells that the Verifiable Presentation is expected in JSON Web Token format and proof type, the signature algorithm, is of type `ES256K`. Throughout this work the `ES256K` is provided with the `secp256k1` curve. The `contraints` field describes one field typed as string: `IDCardCredential`. The field must be present in the `type` field of the Verifiable Credential wrapped in the Verifiable Presentation response. The character '$' indicates the `type` should be found at the root of the Verifiable Credential. The `IDCardCredential` and its values must also be found in the `credentialSubject` of the Verifiable Credential.

Log in to wallet and permissions (Step 3)

The User authenticates and logs in to wallet with, e.g., fingerprint or face recognition. The wallet presents presents information about the attributes OP requires from the User in a Verifiable Presentation. After inspection of requested attributes and the requesting party, user approves, creates a Verifiable Presentation from Verifiable Credentials and responds it to OP.

In detail, the Wallet verifies the signature of the `request` object JWT. This is done by resolving the DID Document and dereferencing the `publicKey` with the information provided in the `kid` field in the `header` of the `request` object. The Wallet acquires the requirements about the Verifiable Credentials from the `presentation_definition` of the `request` object and informs the User about the required information. The user approves sending the required `IDCardCredential` credential to OP. If the required credential is not available in the Wallet, the User can request it from the Issuer.

Federated Authorization Response (Step 4)

The Wallet creates 4.9 Verifiable Presentation Response including an `id_token` (JWT), a `presentation_submission` object and a `vp_token` (JWT) which includes the required `verifiable_credentials` (JWT). The `state` must be included in the response if the OP has sent one and then it has to mach the sent value.

```
https://openid-provider.com:3000/interaction/callback/wallet?
   id_token=...
    &presentation_submission=...
    &vp_token=...
```

```
    &state=YUe87dSq
```

Listing 4.9: Verifiable Presentation Response

The wallet sends the 4.9 Verifiable Presentation Response back to the OpenID Provider's `redirect_uri`.

```
// header
{
  "typ": "JWT",
  "alg": "ES256K",
  "kid": "did:ion:
     EiAZF8EUsRKvkuqQeAvtha8WnxFa2VlK3M7XwJ1EsOfEvA#key-1"
}
// body
{
  "iss": "did:ion:
     EiAZF8EUsRKvkuqQeAvtha8WnxFa2VlK3M7XwJ1EsOfEvA",
  "sub": "did:ion:
     EiAZF8EUsRKvkuqQeAvtha8WnxFa2VlK3M7XwJ1EsOfEvA",
  "aud": "did:ion:EiBr3cl0yOq4TDkQ-AioibD8NF2Miml3BQ-40
     smk5Viu0Q",
  "nonce": "acfdef3D2",
  "iat": 1681061627,
  "nbf": 1681061627
}
```

Listing 4.10: Verifiable Presentation Response Id Token

The 4.10 Verifiable Presentation Response Id Token describes the header and body of the `id_token`. The `id_token` body `iss` and `sub` has to be the same, which is the Wallet DID. The wallet is the issuer, signer and the subject of the `id_token`. The audience, `aud`, is the OpenID Providers DID. There are also the standard expiration and date of issuance fields. The `nonce` must match with the value OP sent with `request` object.

In the header of the `id_token`, the `kid` field contains the DID and fraction which describes the `publicKey` to verify the `id_token` signature.

```
{
  "definition_id": "example_jwt_vc",
```

```
  "id": "unique_jwt_vc_presentation_submission_id",
  "descriptor_map": [{
    "id": "id_credential",
    "path": "$",
    "format": "jwt_vp_json",
    "path_nested": {
      "path": "$.vp.verifiableCredential[0]",
      "format": "jwt_vc_json"
    }
  }]
}
```

Listing 4.11: Verifiable Presentation Submission

The 4.11 Verifiable Presentation Submission defines metadata of the `vp_token`, e.g., where in the `vp_token` can the Credential be found and the format of the Presentation and Credential. The `presentation_submission` has to be a separate parameter and cannot be nested in other response parameters [38].

```
// header
{
  "kid": "did:ion:
    EiAZF8EUsRKvkuqQeAvtha8WnxFa2VlK3M7XwJ1EsOfEvA#key-1",
  "alg": "ES256K",
  "typ": "vp+jwt"
}
// body
{
  "iss": "did:ion:
    EiAZF8EUsRKvkuqQeAvtha8WnxFa2VlK3M7XwJ1EsOfEvA",
  "jti": "did:ion:
    EiAZF8EUsRKvkuqQeAvtha8WnxFa2VlK3M7XwJ1EsOfEvA-cf6bfdb9-
    abde-4377-b67f-4313b60cae92",
  "aud": "did:ion:EiBr3cl0yOq4TDkQ-AioibD8NF2Miml3BQ-40
    smk5ViuOQ",
  "nbf": 1681061627,
  "iat": 1681061627,
```

```
  "nonce": "acfdef3D2",
  "vp": {
    "@context": [
      "https://www.w3.org/2018/credentials/v1",
      "https://www.w3.org/2018/credentials/examples/v1"
    ],
    "type": ["VerifiablePresentation"],
    "verifiableCredential": [
      "..."
    ]
  }
}
```

Listing 4.12: Verifiable Presentation Token

The `vp_token`, 4.12 Verifiable Presentation Token, is a JWT signed by the Wallet. The header has type, `typ`, as the `vp+jwt` which describes the this token as Verifiable Presentation in JWT format. The Verifiable Presentation does not have `proof` field, the signature of the JWT proves the integrity and the issuer of the Verifiable Presentation. The `iss` must be the Wallet DID and the intended audience, `aud`, must be the DID of the OpenID Provider. The `jti` is a JWT identifier, a unique identifier for the token. There must be the `nonce` present with the same value which was sent in `authorization_request` and which is also present in the `id_token`. The not before, `nbf`, value indicates the point in time when the token should become valid. In this case, the `nbf` and the `iat` are the same. The `vp` contains `verifiableCredential` which contains the Verifiable Credential in JWT format.

```
// header
{
  "kid": "did:ion:
     EiBAtbiEe2qtLsa5a9_fgPQDUAtxBKXLvpI6Lvpkdrcobg#key-1",
  "alg": "ES256K",
  "typ": "vc+jwt"
}

// body
{
```

```
  "iss": "did:ion:
     EiBAtbiEe2qtLsa5a9_fgPQDUAtxBKXLvpI6Lvpkdrcobg",
  "nbf": 1681061627,
  "iat": 1681061627,
  "jti": "did:ion:
     EiBAtbiEe2qtLsa5a9_fgPQDUAtxBKXLvpI6Lvpkdrcobg-cf6bfdb9-
     abde-4377-b67f-4313b60cae10",
  "sub": "<Wallet-DID>",
  "vc": {
    "@context": [
      "https://www.w3.org/2018/credentials/v1",
      "https://www.w3.org/2018/credentials/examples/v1"
    ],
    "type": [
      "VerifiableCredential",
      "IDCardCredential"
    ],
    "credentialSubject": {
      "given_name": "Alice",
      "family_name": "Bobson",
      "date_of_birth": "12-08-1979"
    }
  }
}
```


Listing 4.13: Verifiable Credential

The `verifiableCredential`, 4.13 Verifiable Credential, which is inside the `vp_token` is also a JWT. The header has `typ` as `vc+jwt` describing it as Verifiable Credential in JWT format. The `iss` is the DID of the Verifiable Credential Issuer as well as the signature of the `verifiableCredential` JWT. The `sub` must be the Wallet DID, i.e., this credential has been issued for the holder of the Wallet. The `vc` must contain the Verifiable Credential and the claims, `given_name` and `family_name`. These were defined in 4.8 Presentation Definition, which was sent in the `authorization_request`.

The OpenID Provider receives the response and has to do following verifications and validations:

1. **id_token**: Validate the signature of the id_token. The OP has to resolve the Wallet DID Document with the information from the id_token. The iss (issuer) and sub (subject) fields in id_token has to contain the Wallet DID. The OP uses the kid in the token header to resolve the DID Document and dereference the correct public key. With the public key the OP must verify that signature matches for the id_token. This proofs that the id_token is created by the Wallet.

   The aud (audience) must be the OPs DID.

   The mandatory nonce must match with value sent in the authorization_request sent to Wallet.

   The exp (expiration time) is an integer value in seconds, passed from 1970-01-01T0:0:0Z (UTC). The current time value measured the same way must be less than the value of exp. The specification allow some minutes to take some clock skew into an account [5].

   The iat (JWT issued at) has the same format as exp, but the value must not be in the future. Same few minutes bounds are allowed as for exp.

2. **presentation_submission** is a plain JSON object and not a JWT. It must be present and it contains information where to find the credentials withing the vp_token.

3. **vp_token** has to be presesent as a JWT. The vp_token is signed by the Wallet and the verification of the signature happens with the same steps as above with the id_token. The header typ must be vp+jwt. The issuer must be the Wallet DID and aud must be the OP DID. Teh nonce must be present and has to match with the one sent with authorization_request. The not-valid-before, issued-at and expiration date, nbf, iat and exp respectively, must be present. The current time must be compared to those values accordingly and approve or reject if the value does not match with the time constraints. The vp must be present and within the vp, the type must be VerifiablePresentation. The verifiableCredential must be present and has to contain JWT of Verifiable Credential.

4. **verifiableCredential** in JWT format must be found within the vp_token. The verifiableCredential must be signed with the Issuer and the signature verification has to be made the same manner as with the id_token and vp_token. In header, the typ must be vc+jwt. The iss must be the DID of the Issuer and the sub must be the DID of the Wallet. The OpenID Provider will check its policies that if the

credentials issued by this Issuer should be accepted. The not-valid-before and issued-at `nbf`, `iat` respectively, must be present. The `exp` is optional. The `vc` must have all the requested attributes about the User. These were defined in the Presentation Definition, which was sent along the `request` object of `authorization_request`.

5. `state`: If provided in the in the `authorization_request`, the `state` value must match with the sent value of the `state`.

When the Federated Authorization Response from Wallet to OpenId Provider is validated, the OP can store User attributes received in the Verifiable Credential to a session variable to use them at later stage.

The user has now authenticated themselves with Wallet using Verifiable Credentials and Verifiable Presentations. The remaining parts of the flow follows the OpenID Connect and OAuth2 Authorization Code Grant Flow.

Authorization response to Redirect URI (Step 5)

The OP creates a single-use `authorization_code` and sends it to predefined Relying Party's `redirect_uri`. If Relying Party (RP) have sent `state` in the original `authorization_request` from RP to OP, the `state` must match here with the sent value. This is the same phase as it is in Authorization Response to Redirect URI in 4.3.

Token Request (Step 6)

The RP needs to exchange the `authorization_code` to `id_token` in the OP's token endpoint. The RP needs to provide the `client_id` and `client_secret` to OP. These values are pre-configured in the OP and exchanged out-of-band with RP. The RP provides also the `authorization_code`, `redirect_uri` and `code_verifier` if the Proof Key for Code Exchange (PKCE) extension is used. This way the RP proves the OP that is has the correct configuration of the OIDC Client with credentials and it also has the correct single-use `authorization_code`.

Token Response (Step 7)

The OpenID Provider creates an `id_token` with including the attributes in Verifiable Credential. The OP creates a Single Sign-On session for the User. This means that some other Relying Party would be using the same OP, user would be directly logged in to Relying Party without authenticating again. The OP could store the user attributes only

in to session variable so that when the session is removed, e.g., logging out from OP or expire, there would be no information about the user in the OP. This way the OP would not have to store any data about the User in any persistent Storage. This choice is not always possible, because there are some workplace related domains, where some central identity management and profiling is a strict requirement.

When the session is created for the User, the OP returns the `id_token` to the RP. The RP must validate the token and this time the RP uses the `.well-known endpoint` to retrieve the public key from JSON Web Key Set. The RP verifies the OP signature of the `id_token` and is able to show User information in the system. The RP does not resolve any DIDs with this flow. The RP uses OP as a trusted third party to convey identity information between Identity Provider, e.g., Wallet and the Relying Party.

This is not the most ideal flow in terms of Self-Sovereign Identity model. There is a trusted third party in between the RP and the Wallet. However, it could be seen as a stepping stone towards self-sovereignty. It is also practical flow; This way the SIOP Wallet could be integrated in to existing federated solutions, with minimal changes to Relying Parties. The OpenID Providers would have to modify their systems to accommodate them to use Verifiable Credentials, Verifiable Presentations and Verifiable Data Registries for logging in and registration purposes.

4.5 Verifiable Presentation Flow

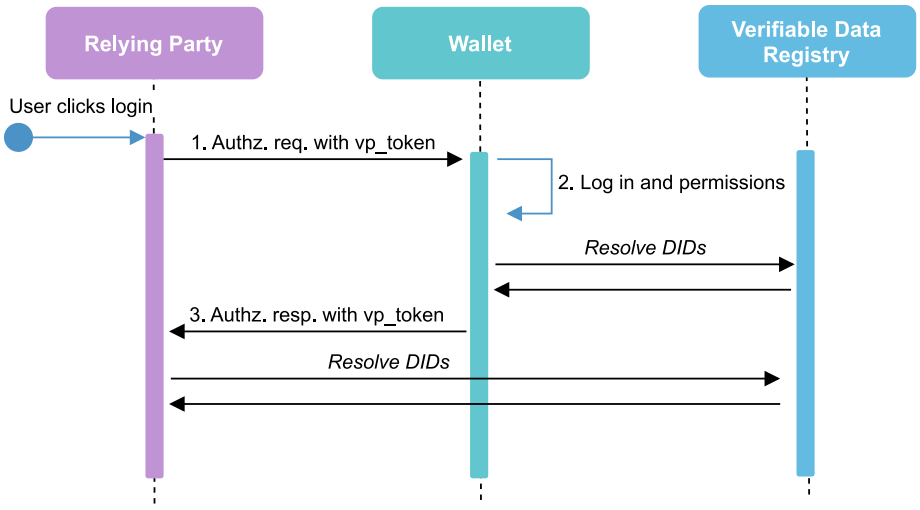

Figure 4.5: Verifiable Presentation Flow

This section describes the 4.5 Verifiable Presentation Flow, which is also a combination

of OpenID Connect, OAuth2 and the draft of OpenID for Verifiable Presentations [38]. This flow contains many of the same components, requirements and parties like Federated Verifiable Presentation Flow 4.4. The most important differentiating factor is that the central OpenID Provider is not present, and the Relying Party and the Wallet communicate directly with each other. The Relying Party acts as a verifier, has to include all the same parameters and verifications as the OpenID Provider had to in Federated Verifiable Presentation Flow 4.4. This flow requires more modifications on the Relying Party in exchange of dropping out the central identity provider. This means direct interactions between service providers and wallets, i.e. users, but also the Single Sign-On feature is also lost. This flow is suitable for different use cases than the Federated Verifiable Presentation Flow 4.4. The implementation of Verifiable Presentation Flow does not support dynamically acquiring the credential, if one is missing from the Wallet. The Verifiable Credential must be imported manually to the Wallet prior for successful authentication.

Authorization Request (Step 1)

This is the same as Federated Authorization Request in Federated Verifiable Presentation Flow. The Relying Party (RP) has to construct the same JWT request object as the OpenID Provider had to in 4.6 Federated Authorization Request Example. The `kid` in the header must point to key in RP's DID document, `client_id` has to be RP's DID and also the `redirect_uri` must be a value that the RP is hosting.

Log in to wallet and permissions (Step 2)

The User authenticates and logs in to wallet. The wallet presents presents information about the attributes Relying Party (RP) requires from the User in a Verifiable Presentation. The Wallet resolves the RP's DID and verifies the JWT signature by resolving the DID Document and using the public key in the document. The Wallet does exactly the same verifications and steps as in Federated Verifiable Presentation Flow. Only the parts where OpenID Provider was involved, has to be exchanged pointing to Relying Party.

Authorization Response (Step 3)

Wallet sends an authorization response to Relying Party (RP) `redirect_uri` with similar payload as in Federated Authorization Response. The `aud` of the `id_token` and the `vp_token` has to be DID of the RP. The Relying Party has to do the same checks, DID resolving and verifications as the OpenID Provider had in case of the Federated Authoriza-

tion Response. The Relying Party also checks it's policies that the Issuer of the Verifiable Credentials is an approved issuer.

When the payload is verified, the RP can initiate the authenticated session with the User, i.e., the User is now logged in to RP using Verifiable Credentials and Verifiable Presentation.

The Verifiable Presentation Flow does not require OpenID Provider in between of the Relying Party and the Wallet. However, the Relying Party is now also a Verifier which requires the Relying Party to be more involved in the process. The Relying Party has to verify the payloads, signatures and resolve DIDs. Because there is no central OpenID Provider, Single Sign-On feature cannot be used. Every session is between the Wallet and the every Relying Party.

5 Discussion

The OpenID4VC [40] works when the OpenID Connect is enriched with the Verifiable Credentials, Verifiable Presentations and Decentralized Identifiers (DIDs). The Self-Issued OpenId Provider Wallet is also able to support and store the Verifiable Credentials. The components developed in the study are not production-level software, i.e., there are some shortcomings, but they serve their purpose for this work. The pros, cons and other observations are discussed in this chapter.

The resolver which the provided Issuer, Relying Party, Wallet and OpenID Provider, are using in this study, comes directly from Identity Overlay Network (ION) Tools [41]. The resolver is able to resolve only the `did:ion` type of Decentralized Identifiers. This is sufficient for this study, but in production the resolver should be more of a universal type. Every component resolving DIDs should implement their own resolver rather than rely solutions of other developers because of security, privacy and trust reasons. However, for development purposes online universal resolvers, like one from Decentralized Identity Foundation [47] are user-friendly. They can be used for confirming if a DID is published in its blockchain or not. The ION Tools proved to be very useful for this study. There is one caveat which would prevent using ION Tools in certain use cases. The time to register the DID did take at least two hours and sometimes much longer. If the use case requires faster registration, then other solution for managing the DIDs should be used. For this feasibility study the DID registration time did not matter. The DIDs were created and registered separately and the OpenID4VC does not register any DIDs when used. The method used to create and register the DIDs can be found in 7.8 in the Appendix. It was preferable for the study that anyone who wants to repeat the steps taken in the study, is able to do so without registering to any service and there is no need to purchase any subscriptions to enable components or APIs.

The signatures of the JSON Web Tokens (JWTs) are almost exclusively done with the ION Tools and with an EC key pair on the *secp256k1* curve, with algorithm *ES256K*. The ION Tools support currently only two key types, i.e., with curves *secp256k1* and *Ed25519*. The only exception in the system is the the last token reponse from OpenID Provider in the 4.5 Verifiable Presentation Flow, where the `id_token` is signed with algorithm *RSA-SHA256*. The signature verification is done with the jose-library [48], because of wider support for

signature algorithms compared to ION Tools.

The items stored in the blockchain with ION Tools are loosely defined. There are service endpoints in the DID documents, e.g., `https://relying-party.com:3001` and `https://openid-provider.com:3000`, and the URLs are just set in the DID Document as values. There is no way to check whether the controller of such domains has set those values or someone else. The domains in the study are not resolvable at all without modifying the `/etc/hosts` file of the host machine and mapping the domains explicitly to other location, e.g., *localhost*. The domain ownership proving problem could be solved, e.g., by publishing a `web:did` in the domain's `well-known/did-configuration` and linking it with the DID document.

All of the components of the OpenID4VC, the OpenID Provider, the Issuer, the Wallet and the Relying Party are using self-signed certificates for TLS. All the certificates and private keys are stored in the software repository of OpenID4VC [40] instead of following best practices. All of the keys are in clear text and effortlessly available along with the other files. This was an intentional decision to make technical part more accessible and remove the waiting time of publishing DIDs. Normally private keys should never be published in the repositories. The private keys for update and recover the DID document are left out. So the private key is valid for signing messages under DID in question, but the DID document can not be changed. There is a method 7.8 In the Appendix, which describes how to generate DID with the ION Tools.

The session handling, input validation and API protections are also lacking. The user interfaces in the OpenID4VC are also not adequate for long term use, but rather quickly done for enabling the use of components. These shortcuts have been made in order to direct the time and resources on the actual focus of the study, but they can be seen as areas of improvement if some parts of the architecture, components or ideas are developed further in industry or research.

The Verifiable Credentials and Verifiable Presentations do not have a `proof` field, instead the signature of the JWT proves the integrity [46] of the Verifiable Credential and Presentation. The Verifiable Credentials and Verifiable Presentations in requests are sent as JSON Web Tokens (JWTs). The JWT signatures are the integrity protection mechanisms for both, Verifiable Credentials and Verifiable Presentations.

The Relying Party constructs and sends requests and handles responses from the OpenID Provider and from the Wallet. Relying Party has one OpenID Connect Client configured with OpenID Provider and the OpenID Connect Client credentials, id and secret, are hard

coded in to the applications. Normally the OpenID Connect Client credentials are not handled this way and never stored in code repositories.

Relying Party is able to use federated authentication through OpenID Provider and also authenticate directly with the Wallet. The Relying Party is able to display to user information on successful authorization flow. The Relying Party can also the verify signature of the given `id_token`, handle local authenticated user sessions, request Verifiable Credentials with OpenID Connect and display user information. Relying Party could be as any service provider which requires user to log in. For the OpenID4VC, the only Verifiable Credential it is directly accepting from Wallet is of type `IDCardCredential`, but adding or changing the credential type would typically require a moderate implementation effort. The amount of implementation effort depends of the type of the credential, how many credentials there would be and from how many different issuers.

The Wallet has no proper database, authentication and key management. But it serves the purpose in storing, listing and removing, and releasing Verifiable Credentials on request to other applications. The Wallet supports only dynamic client registration, i.e., the OpenID Clients are recognized and accepted when they pass their DIDs in the request objects. The Wallet also returns an error to the requestor's `redirect_uri` if the request for Verifiable Presentation is made and there are no credentials in the Wallet. For better user experience the acquisition of credentials could start directly from the Wallet. The wallet also rejects Verifiable Credential issued to some other DID than the Wallet's own.

The choice of producing own web based application for the Wallet was made in favor over using available native applications. One reason for the choice was that the tested native applications had problems to interact with each other. They did not always accept credentials from each other's systems. It was better to have one simple web application for total control of requests, responses and integrations with other applications. For further development better choice could be a mobile application and preferably one which would accept the Verifiable Credentials for OpenID Connect [37].

The European Digital Identity Wallet legislation aims [49] [50] to improve cross-border electronic identification and trust services. The legislation requires each member state to provide its citizens a digital wallet application. At first the wallet application would host digital identity card of its owner and the digital identity card would be valid in all member states. The functionalities and markets for digital wallets are set to increase in the future. The wallets are estimated to be used, e.g., for digitally signing documents, act on behalf of another person or a company and to hold a passport. Digital And Population

Data Services Agency of Finland with Ministry of Finance of Finland are responsible of producing the national digital identity wallet for Finland [51].

The OpenID Provider of the OpenID4VC [40] makes it almost compatible with the Finland national Wallet [51]. A few changes would be needed to achieve full compatibility, but they are estimated to be minor ones. The OpenID4VC would have to increase options for the authorization requests and add support for new DID methods. The changes include: Authorization requests would have to be unsigned, registration URI query parameter should be used instead of request object, the `did:web` support should be added and used instead of the `did:ion`.

In May 2023, the Finnish Wallet implementation project is halted because of government changes which means that the legislative discussion can not be held in parliament [52].

Normally the Issuer should verify the identity of the subject of the credential, e.g. the Wallet. In OpenID4VC the Issuer authentication in bypassed. The Issuer grants `IDCardCredential` Verifiable Credential with `given_name`, `family_name` and `date_of_birth` attributes. There is also a mandatory field for the DID of the subject, i.e., the Wallet. The attributes can be chosen freely in the implementation, but normally there would be an authentication process, e.g., OpenID Connect Authorization Code Grant Flow, to verify the identity of the requesting party. The Issuer signs the Verifiable Credential with its private key.

The OpenID Connect for Verifiable Presentations and Verifiable Credentials in federated login has been added to the OpenID Provider (OP). The OP is still able to authenticate with username and password using 4.3 Authorization Code Grant Flow, but there is an option to use the Wallet as a federated password-free login method, i.e., 4.4 Federated Verifiable Presentation Flow. The Single Sign-On works also with the new federated method. The policies for the acceptable Verifiable Credentials have been included to the OpenID Provider. The policies are simple. There is an array of DIDs of the trusted credential issuers. If the Issuer of the credential in the Verifiable Presentation is not found in the OP's trusted list, the Verifiable Presentation is rejected and the authorization flow is aborted. This check occurs very late in the authorization flow and an improvement would be to inform the User before the federated authorization request is sent from the OP.

The Verifiable Credentials and Presentations in the OpenID4VC are small in size. The `IDCardCredential` has only three attributes and the Verifiable Presentation contains always only one item. The specifications for the Wallet [39] recommend that the authoriza-

tion URL should not exceed 2048 bytes. The limit can be reached when more and more credentials are added to the request. Large requests could get blocked by firewalls. Also if OpenID Provider would construct the large authorization request as a QR code for native applications, the QR code might become too complex to decipher with smartphones.

The OpenID4VC does not support selective disclosure of credential attributes, which should be for better privacy. Now the OP requests the whole `IDCardCredential` and on user's permission the Wallet releases it. The OpenID4VC does not utilize Zero Knowledge Proofs either. Further study is needed to find out how they could be included in to this implementation, or in a similar solution.

6 Conclusions

This study assessed the feasibility of using Verifiable Credentials and Verifiable Presentations with OpenID Connect (OIDC). The OIDC was enhanced to be able to authenticate directly with Self-Issued OpenID Wallet and using federated method, through an OpenID Provider. The study presented two new flows to enrich the OIDC: Verifiable Presentation Flow and Federated Verifiable Presentation Flow. The implementation, the OpenID4VC [40], includes the modified OpenID Provider, a web based credential Wallet, an Issuer for Verifiable Credentials and a Relying Party which consumes the Verifiable Credentials. The OpenID4VC is a working system where the Verifiable Credentials can be used for authentication. The User can approve or reject the credential release from Wallet to requesting party. There is room for improvement in the fine-grained user attribute release, i.e., selective disclosure. That is, e.g., on age related matter, if the user is over 18 years of age or not instead of proper date of birth.

The authentication could be completely password-free. If the Wallet application would utilise biometric authentication method, e.g., fingerprint or face recognition. Also if the Wallet would already have the required credential in the Wallet. In case the credential would not be in the Wallet, the Issuer would utilise biometric authentication method as well as the Wallet. This study did not cover authentication in the Wallet or the Issuer.

The modifications to support Verifiable Presentations and Verifiable Credentials in other OpenID Providers could be generalized. The OpenID Providers would have to be able to: create presentation submissions, sign and verify JWTs with algorithms in common with the wallets, host policies for accepted credential issuers, able to resolve preferably many different types of DIDs and verify and validate payloads the OP receives from wallets.

If a relying party, which would have already a working OpenID Connect authorization flow configured with an OpenID provider, would not have to do any changes to codebase or configuration for the new Federated Verifiable Presentation Flow to work. The Single Sign-On works with other relying parties using the same OpenID provider, e.g., school or work environments.

The Verifiable Presentation flow does not have OpenID Provider between the Wallet and the Relying Party. The User authenticates directly to Relying Party using their credential Wallet. The Verifiable Presentation flow requires the Relying Party to modify its code

base and configurations in exchange of direct exchange of credentials. There is no Single Sign-On in this flow, i.e., authenticating with another Relying Party requires requests to Wallet, authenticating and approving the release of credentials.

The OpenID Connect is widely used in various systems incorporating authentication mechanisms. Including Verifiable Credentials into OpenID Connect offers one option more for users to sign in securely. It is essential that all the mechanisms are well known and audited. Authentication is very central functionality and gatekeeper to various systems. There are many opportunities for research and industry.

The Verifiable Credentials and their use opens up multiple possibilities across different industries and other areas, e.g., travel, study, banking, game industry and research. The European Union has the identity wallet (EUDI Wallet) initiative and it is yet to be seen that how different areas of business can integrate to the coming identity wallets, even from private sector. The EUDI wallet is to initially help people to relocate within EU and even outside EU and use digital services in any EU country with high assurance.

This study could be extended for further research and assessment of the security of the OpenID Connect with Verifiable Credentials and Presentiation in different identity systems. One potential area for research could be incorporating Verifiable Credentials and Presentations into Financial Grade OpenID Connect (FAPI) protocol. Other potential extension to this study would be the use of encrypted JSON Web Tokens (JWEs) instead to incorporate message level encryption in the requests. Already the Wallet in this study is digitally signing the JSON Web Token (JWT) payloads and the Wallet could be improved to sign documents of agreed formats.

The Verifiable Credentials, Verifiable Presentations, Verifiable Data Registry and digital Wallets, all of which are part of the Self-Sovereign Identity model, have shown to be useful where verifiability and non-repudiability is a requirement. There is still work to do to protect the privacy of the user and give the user more control over their attributes shared in systems. The Verifiable Credentials and Presentation can be seen as a checkpoint moving towards Self-Sovereign Identity. If the Self-Sovereign Identity would be feasible goal at all. There will always be some authorities granting permissions and credentials, which will not be given for user's total control, e.g., official documents issued by government. The practical model [3] seems more reasonable, where there are mixed decentralized and centralised domains. The term Self-Sovereign Identity might need some rework, but the pursue of the Self-Sovereign Identity has brought interesting ideas and implementations for enhancing the fluency, pace and security of digital services.

7 Appendix

The Appendix includes some of the important methods of the implementation.

The 7.1 takes the Verifiable Credential as JWT and the Wallet DID as parameter and returns a boolean for the validity of the Verifiable Credential.

```
const isValidCredJwt = (jwt, walletDID) =>  {
  const header = JSON.parse(Buffer.from(jwt.split(".")[0], '
     base64').toString('utf8'));
  const body = JSON.parse(Buffer.from(jwt.split(".")[1], '
     base64').toString('utf8'));
  const keyId = header.kid;
  const did = keyId.split("#")[0];
  if(did != body.iss) {
    console.log("issuer and did in header are not the same");
    return false;
  }
  if(!isValidDate(body)) {
    console.log("token is not yet active, expired or contains
        otherwise false time signature.");
    return false;
  }
  if(body.sub != walletDID) {
    console.log("Credential subject mismatch from wallet");
    return false;
  }
  console.log("Valid DID JWT")
  return true;
};6
```

Listing 7.1: isValidCred()

The 7.2 takes in the JWT as parameter parses the public key id and the algorithm from the header of the JWT. Then it resolves the DID to a DID Document and resolves the correct public key. The algorithm, the JWT and the public key are then sent to 7.6. The

boolean response is returned as result.

```
const parseJwt = async (jwt) => {
  const header = JSON.parse(Buffer.from(jwt.split(".")[0], '
     base64').toString('utf8'));
  const body = JSON.parse(Buffer.from(jwt.split(".")[1], '
     base64').toString('utf8'));
  const kid = header.kid;
  let doc = undefined;
  let publicJwk;
  let validSignature = false;
  if(kid.indexOf("did:") >= 0) {
    // public key needs to be aquired from did document
    const did = kid.split("#")[0];
    const keyId = `#${kid.split("#")[1]}`;
    doc = await resolve(did);
    publicJwk = getPublicKeyFromDocument(doc, keyId);
    validSignature = await isValidSignature(header.alg, jwt,
       publicJwk);
    console.log("is valid signature: " + validSignature);
    return validSignature;
  } else {
    return false;
  }
};
```

Listing 7.2: parseJwt()

The 7.3 is a helper methods which returns the current date and time in seconds elapsed since 00:00:00 on 1st of January 1970, i.e., the Unix time.

```
const getDateEpoch = () => {
  const now = new Date()
  const utcMilllisecondsSinceEpoch = now.getTime() + (now.
     getTimezoneOffset() * 60 * 1000)
  const utcSecondsSinceEpoch = Math.round(
     utcMilllisecondsSinceEpoch / 1000)
  return utcSecondsSinceEpoch;
```

```
};
```

Listing 7.3: getDateEpoch()

The 7.4 is a helper methods which checks the JWT date fields `exp`, `nbf` and `iat`.

```
const isValidDate = (body) => {
  const exp = body.exp || undefined;
  const now = getDateEpoch();
  if(exp) {
    return (now >= body.iat) &&
  (now >=body.nbf) && (now <= exp);
  } else {
    return (now >= body.iat) &&
    (now >=body.nbf);
  }
};
```

Listing 7.4: isValidDate()

The 7.5 retrieves the public key from the DID Document. The method takes in the DID Document and the key identifier as parameters. The method returns the public key in JWK format or an empty object accordingly.

```
const getPublicKeyFromDocument = (doc, keyId) => {
  console.log("public key from did document method")
  const verificationMethod = doc.didDocument.
     verificationMethod;
  const keyResult = verificationMethod.filter(key => (key.id
     == keyId) || (key.id == `#${keyId}`)).pop();
  if(keyResult) {
    // key found
    const publicJwk = keyResult.publicKeyJwk;
    return publicJwk;
  } else {
    // key not found
    console.log("key not found");
    return {};
  }
```

```
};
```

Listing 7.5: getPublicKeyFromDocument()

The 7.6 verifies the signature of the given JWT. The method also requires the algorithm and the public key of the signature as parameters. The verification is done with the `jose` library. The method returns a boolean according to outcome of signature verification.

```
const isValidSignature = async (alg, jwt, publicJwk) => {
  const publicKey = await jose.importJWK(publicJwk, alg);
  let verSig = undefined;
  let isValid = false;
  if(alg.toLowerCase() == "none") {
    return isValid;
  }
  try {
    verSig = await jose.compactVerify(jwt, publicKey)
    if(verSig) {
      isValid = true;
    }
  } catch(error) {
    isValid = false;
    console.log(error)
  }
  return isValid;
};
```

Listing 7.6: isValidSignature()

The 7.7 is a method which is used to create a request object which is sent to wallet. The method takes in state and nonce as parameters. The `ID` is a global variable for the Decentralized Identifier of the solution. The `URI` is a global variable for the URI of the solution.

The method creates a JWT and signs with its private key. This method also includes the `presentation_definition`, which informs the Wallet about the Verifiable Credential required in response.

```
const createRequestObject = async (state, nonce) => {
```

```
const privateJwk  = JSON.parse(fs.readFileSync('did-private
   .json', 'utf8'));
const alg = "ES256K";
const iat = getDateEpoch();
const header = {
  "kid": `${ID}#key-1`,
  "alg": alg,
  "typ":"oauth-authz-req+jwt"
}
const body = {
  "client_id": `${ID}`,
  "client_id_scheme": "did",
  "iss": ID,
  "nbf": iat,
  "iat": iat,
  "jti": `${ID}-${uuidv4()}`,
  "response_type": "id_token vp_token",
  "redirect_uri": `${URI}/redirect`,
  "nonce": nonce,
  "scope": "openid",
  "state" :state,
  "presentation_definition": {
    "id": uuidv4(),
    "input_descriptors": [{
      "id": uuidv4(),
      "format": {
        "jwt_vp": {
          "alg": [
            "ES256K"
          ]
        }
      },
      "constraints": {
        "fields": [{
          "path": ["$.type"],
```

```
            "filter": {
              "type": "string",
              "pattern": "IDCardCredential"
            }
          }]
        }
      }]
    }
  }
  const jwt = await sign({ header: header, payload: body,
     privateJwk });
  return jwt;
};
```

Listing 7.7: createRequestObject()

The 7.8 is a method which can be used to create a DID, a DID Document and publish, i.e. anchor, it to blockchain. It takes only few seconds to run, but the publishing of the DID might take at least two hours. The method creates a new key pair with private key and corresponding public key. The method sets created public key in `publicKeys` and 'https://example.com:3008' in `serviceEndpoint`. The DID is then created, DID Document is published, i.e., *anchored* and the method writes four files on the same directory where the hosting file of the method is located. The files are `did-info.json`, `did-private.json`, `did-public.json` and `ion-did-ops-v1.json`.

The `did-info.json` contains the long and short form of the DID created. The short form is the one that is used in the implementations and once the DID is published the short form is available to use. The `did-private.json` and `did-public.json` contains the private and the public keys. The public key is published in the DID Document. The `ion-did-ops-v1.json` file contains the private keys for updating and recovering the DID document. The update and recovery is not within the scope of the study.

```
const { sign, verify, anchor, DID, generateKeyPair, resolve }
    = require('@decentralized-identity/ion-tools');
const fs = require('fs')
const { writeFile }  = require('fs/promises');

const anchorDID = async () => {
```

```
let authnKeys = await generateKeyPair();
let did = new DID({
  content: {
    publicKeys: [
      {
        id: 'key-1',
        type: 'EcdsaSecp256k1VerificationKey2019',
        publicKeyJwk: authnKeys.publicJwk,
        purposes: [ 'authentication','assertionMethod', '
           capabilityInvocation', 'capabilityDelegation' ]
      }
    ],
    services: [
      {
        id: 'domain-1',
        type: 'LinkedDomains',
        serviceEndpoint: 'https://example.com:3008'
      }
    ]
  }
});

let longFormDID = await did.getURI();
let shortFormDID = await did.getURI('short');

// get keys for modify and recover
let ionOps = await did.getAllOperations();

let createRequest = await did.generateRequest();
let anchorResponse = await anchor(createRequest);

console.log(anchorResponse);
// store the DIDs (short and long) to a file
await writeFile('./did-info.json', JSON.stringify({"short":
   shortFormDID,"long":longFormDID}));
```

```
  // store the keys for modify and recover to a file
  await writeFile('./ion-did-ops-v1.json', JSON.stringify({
     ops: ionOps }));
  // store the private key to a file
  await writeFile('./did-private.json', JSON.stringify(
     authnKeys.privateJwk));
  // store the public key to a file
  await writeFile('./did-public.json', JSON.stringify(
     authnKeys.publicJwk));
};
```

Listing 7.8: anchorDID()

The 7.9 describes the approach, which the OpenID Provider and the Relying Party is handling their policies regarding if the issuer in question is trusted or not. Trusted Issuer DIDs are inserted in to an array, `trustedIssuerDID`, in `vcIssuerPolicies` object. If the issuer given as parameter to method isTrustedIssuer(), is found in the *trustedIssuerDID*, the issuer can be trusted. There is also a `strict` field which can be used for development purposes. If the `strict` is false, then no further checks are made and the issuer is regarded as trusted, like all the issuers.

```
const vcIssuerPolicies = {
  strict: true,
  trustedIssuerDID : ["did:ion:
     EiBAtbiEe2qtLsa5a9_fgPQDUAtxBKXLvpI6Lvpkdrcobg"]
};
...
const isTrustedIssuer = (iss) => {
  const strict = vcIssuerPolicies.strict;
  const trustedIssuerDID = vcIssuerPolicies.trustedIssuerDID;
  // if no issuer checking
  if(!strict) {
    console.log("strict is false: no issuer checking");
    return true;
  } else {
    const trusted = trustedIssuerDID.includes(iss);
    console.log(`The Verifiable Credential Issuer (DID: ${iss
```

```
      }) is trusted: ${trusted}`);
    return trusted;
  }
};
```

Listing 7.9: isTrustedIssuer()

References

[1] J. Isaak and M. J. Hanna. "User Data Privacy: Facebook, Cambridge Analytica, and Privacy Protection". In: *Computer* 51.8 (2018), pp. 56–59. DOI: 10.1109/MC.2018.3191268.

[2] M. Ferdous et al. "User-controlled identity management systems using mobile devices". PhD thesis. University of Glasgow, 2015.

[3] M. S. Ferdous, F. Chowdhury, and M. O. Alassafi. "In Search of Self-Sovereign Identity Leveraging Blockchain Technology". In: *IEEE Access* 7 (2019), pp. 103059–103079. DOI: 10.1109/ACCESS.2019.2931173.

[4] S. Cantor, J. Kemp, R. Philpott, and E. Maler. *Assertions and Protocols for the OASISSecurity Assertion Markup Language(SAML) V2.0*. 2005. URL: https://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf (visited on 05/16/2022).

[5] F. N. Sakimura, NRI, J. Bradley, P. Identity, M. Jones, Microsoft, B. de Medeiros, Google, C. Mortimore, and Salesforce. *OpenID Connect Core 1.0 incorporating errata set 1*. 2014. URL: https://openid.net/specs/openid-connect-core-1_0.html (visited on 05/16/2022).

[6] I. E. T. Force. *The OAuth 2.0 Authorization Framework*. RFC 6749. Microsoft, Oct. 2012, pp. 1–75. URL: https://www.rfc-editor.org/rfc/rfc6749.

[7] C. Allen. *The Path to Self-Sovereign Identity*. 2016. URL: https://www.lifewithalacrity.com/2016/04/the-path-to-self-soverereign-identity.html (visited on 05/16/2022).

[8] A. Mühle, A. Grüner, T. Gayvoronskaya, and C. Meinel. "A survey on essential components of a self-sovereign identity". In: *Computer Science Review* 30 (2018), pp. 80–86. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2018.10.002. URL: https://www.sciencedirect.com/science/article/pii/S1574013718301217.

[9] U. Der, S. Jähnichen, and J. Sürmeli. "Self-sovereign Identity – Opportunities and Challenges for the Digital Revolution". In: (Dec. 2017).

[10] P. Grassi, M. Garcia, and J. Fenton. *Digital identity guidelines*. Tech. rep. National Institute of Standards and Technology, 2020.

[11] J. Bernal Bernabe, J. L. Canovas, J. L. Hernandez-Ramos, R. Torres Moreno, and A. Skarmeta. "Privacy-Preserving Solutions for Blockchain: Review and Challenges". In: *IEEE Access* 7 (2019), pp. 164908–164940. DOI: 10.1109/ACCESS.2019.2950872.

[12] T. E. Comission. *General Data Protection Regulation - GDPR*. 2018. URL: https://gdpr-info.eu (visited on 05/16/2022).

[13] V. One. *A Globally Interoperable Network for Identity*. 2023. URL: https://veres.one (visited on 06/01/2023).

[14] Jolocom. *We create solutions for the future of digital identity*. 2023. URL: https://jolocom.io (visited on 06/01/2023).

[15] S. Foundation. *Sovrin*. 2023. URL: https://sovrin.org (visited on 06/01/2023).

[16] M. Security. *Microsoft Entra Verified ID*. 2023. URL: https://www.microsoft.com/en-us/security/business/identity-access/microsoft-entra-verified-id (visited on 06/01/2023).

[17] M. Security. *Decentralized identity and verifiable credentials*. 2023. URL: https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RE5cxkr?culture=en-us&country=us (visited on 06/01/2023).

[18] Veramo. *Performant and modular APIs for Verifiable Data and SSI*. 2023. URL: https://veramo.io (visited on 06/01/2023).

[19] uPort. *uPort has evolved*. 2023. URL: https://www.uport.me (visited on 06/01/2023).

[20] Serto. *Trust with control*. 2023. URL: https://www.serto.id (visited on 06/01/2023).

[21] E. Comission. *European Blockchain Services Infrastructure (EBSI) Blockchain*. 2023. URL: https://ec.europa.eu/digital-building-blocks/wikis/display/EBSI/Home (visited on 05/16/2023).

[22] E. C. eIDAS Expert Group. *European Digital Identity Architecture and Reference Framework*. 2023. URL: https://digital-strategy.ec.europa.eu/en/library/european-digital-identity-wallet-architecture-and-reference-framework (visited on 05/16/2023).

[23] World Wide Web Consortium (W3C). *Verifiable Credentials Data Model v1.1*. 2022. URL: https://www.w3.org/TR/vc-data-model/ (visited on 05/16/2022).

[24] I. E. T. Force. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 7159. Google, Inc., Mar. 2014, pp. 1–15. URL: https://www.rfc-editor.org/rfc/rfc7159.

[25] World Wide Web Consortium (W3C). *JSON-LD 1.1, A JSON-based Serialization for Linked Data*. 2020. URL: https://www.w3.org/TR/json-ld11/#data-model (visited on 05/16/2022).

[26] I. E. T. Force. *JSON Web Token (JWT)*. RFC 7519. Microsoft, Ping Identity, NRI, May 2015, pp. 1–29. URL: https://www.rfc-editor.org/rfc/rfc7519.

[27] World Wide Web Consortium (W3C). *W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes*. 2012. URL: https://www.w3.org/TR/xmlschema11-2/#dateTime (visited on 05/16/2022).

[28] I. E. T. Force. *JSON Web Signature (JWS)*. RFC 7515. Microsoft, Ping Identity, NRI, May 2015, pp. 1–59. URL: https://www.rfc-editor.org/rfc/rfc7515.

[29] World Wide Web Consortium (W3C). *Verifiable Credential Data Integrity 1.0 - Securing the Integrity of Verifiable Credential Data*. 2022. URL: https://w3c.github.io/vc-data-integrity/ (visited on 05/16/2022).

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

[31] World Wide Web Consortium (W3C), Manu Sporny (Digital Bazaar) Dmitri Zagidulin (DCC (MIT Digital Credentials Consortium)). *EdDSA Cryptosuite v2020*. 2022. URL: https://www.w3.org/community/reports/credentials/CG-FINAL-di-eddsa-2020-20220724/ (visited on 05/16/2022).

[32] World Wide Web Consortium (W3C). *Decentralized Identifiers (DIDs) v1.0*. 2022. URL: https://www.w3.org/TR/did-core/ (visited on 05/16/2022).

[33] Z. A. Lux, D. Thatmann, S. Zickau, and F. Beierle. "Distributed-ledger-based authentication with decentralized identifiers and verifiable credentials". In: *2020 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*. IEEE. 2020, pp. 71–78.

[34] World Wide Web Consortium (W3C). *DID Specification Registries*. 2023. URL: https://www.w3.org/TR/did-spec-registries/ (visited on 05/16/2023).

[35] World Wide Web Consortium (W3C). *The did:key Method v0.7*. 2022. URL: https://w3c-ccg.github.io/did-method-key (visited on 05/16/2023).

[36] B. Alzahrani. "An information-centric networking based registry for decentralized identifiers and verifiable credentials". In: *IEEE Access* 8 (2020), pp. 137198–137208.

[37] T. Lodderstedt, K. Yasuda, T. Looker. *OpenID for Verifiable Credential Issuance*. 2022. URL: https://openid.net/specs/openid-4-verifiable-credential-issuance-1_0.html (visited on 05/16/2023).

[38] O. Terbu, T. Lodderstedt, K. Yasuda, A. Lemmon, T. Looker. *OpenID for Verifiable Presentations*. 2022. URL: https://openid.net/specs/openid-4-verifiable-presentations-1_0.html (visited on 05/16/2023).

[39] K. Yasuda, M. Jones, T. Lodderstedt. *Self-Issued OpenID Provider v2*. 2023. URL: https://openid.net/specs/openid-connect-self-issued-v2-1_0.html (visited on 05/16/2023).

[40] Miika Pärni. *OIDC4VC*. 2023. URL: https://github.com/mcparni/OpenID4VC (visited on 05/16/2023).

[41] Decentralized Identity Foundation. *ION Tools*. 2023. URL: https://github.com/decentralized-identity/ion-tools (visited on 05/16/2023).

[42] Philip Skokan. *oidc-provider*. 2023. URL: https://github.com/panva/node-oidc-provider (visited on 05/16/2023).

[43] Decentralized Identity Foundation. *ION Tools*. 2023. URL: https://identity.foundation/ion/ (visited on 05/16/2023).

[44] I. E. T. Force. *OAuth 2.0 Authorization Server Metadata*. RFC 8414. June 2018, pp. 1–22. URL: https://www.rfc-editor.org/rfc/rfc8414.txt.

[45] R. Hedberg, Ed., M.B. Jones, A.Å. Solberg, J. Bradley, G. De Marco, V. Dzhuvinov. *OpenID Connect Federation 1.0*. 2023. URL: https://openid.net/specs/openid-connect-federation-1_0.html (visited on 05/16/2023).

[46] World Wide Web Consortium (W3C). *Securing Verifiable Credentials using JSON Web Tokens*. 2023. URL: https://w3c.github.io/vc-jwt/ (visited on 05/16/2023).

[47] Decentralized Identity Foundation. *DIF Universal Resolver*. 2023. URL: https://dev.uniresolver.io/ (visited on 05/16/2023).

[48] Philip Skokan. *Jose*. 2023. URL: https://github.com/panva/jose (visited on 05/16/2023).

[49] F. Digital and P. D. S. Agency. *European wallet application*. 2023. URL: https://dvv.fi/en/digital-identity-reform (visited on 05/16/2023).

[50] M. of Finance Finland. *European wallet application*. 2023. URL: https://vm.fi/en/european-wallet-application (visited on 05/16/2023).

[51] Finnish Digital And Population Data Services Agency. *SIOPv2 POC - Guide for Relying Parties*. 2022. URL: https://wiki.dvv.fi/display/DHHJD/SIOPv2+POC+-+Guide+for+Relying+Parties (visited on 05/16/2023).

[52] F. Digital and P. D. S. Agency. *Digital identity reform interrupted, need for digital identity solutions did not disappear*. 2023. URL: https://dvv.fi/-/digitaalisen-henkilollisyyden-uudistus-keskeytyi?languageId=en_US (visited on 05/16/2023).