POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea

Study of authentication models and implementation of a prototype by using eID and Distributed Ledger Technologies

Relatori
prof. Antonio Lioy
prof. Diana Gratiela Berbecaru

Candidato
Mattia Morosi

Anno accademico 2021-2022

Summary

The aim of this thesis project is to analyse the existing Self-Sovereign Identity (SSI) and Digital Wallet solutions used to manage digital information which identifies a user, to implement a prototype of an infrastructure simulating the issuance and usage of a digital identity called ValID. This concept spreads in the last two years due to the pandemic of COVID-19 because of the impossibility of performing in-person activities. For this reason, companies needed to move their services toward the internet, and citizens needed a way to demonstrate their identity or something related to it. Mainly three identity management (IDM) systems have been developed through the years. The reference model of this project is that of the Self-Sovereign Identity because of its principle of giving the user sovereignty over his personal information. To develop this infrastructure, a study of standards, protocols, the eDIAS regulation, and of Distributed Ledger Technologies (DLTs) has been performed. The IOTA Tangle has proved to be the more suitable DLT for ValID, due to its scalability with respect to blockchain solutions. This is mainly related to the loading of data. In blockchain solutions, transactions are inserted in a block that is linked to the previous one to build a chain. In a system based on the IOTA Tangle, instead, a block is linked to eight previous blocks. ValID leverages on the IOTA mainnet, a network based on the IOTA Tangle, to store Decentralized Identifiers (DIDs) and DID Documents representing the users. Following the SSI principles, the Verifiable Credentials (VCs) that the user obtains from the attribute provider are not loaded on the IOTA mainnet but they are stored locally on the user's digital wallet. ValID is composed of three modules, which are an Identity Provider (IdP), which includes also an attribute provider, a verifier, and the digital wallet of the user. The communication among them uses HTTPS to improve security. Giving sovereignty over his personal information to the user is an advantage having him the possibility to decide when and with whom to share it without relying on third-party intermediaries. This could be the beginning of a new era for security and data protection in the digital world.

Contents

1 Introduction to Digital Identity and Self-Sovereign Identity 6

2 Study of the standards (protocols) and specifications 8

2.1 W3C specifications . . . 8

2.1.1 Decentralized Identifier . . . 8

2.1.2 Verifiable Credential . . . 10

2.2 eIDAS . . . 14

2.2.1 eIDAS Revision . . . 17

2.3 General Data Protection Regulation (GDPR) . . . 17

2.4 SAML 2.0 . . . 19

3 What is a digital wallet? 23

3.1 Connect.me . . . 23

3.2 Jolocom . . . 23

3.3 Trinsic . . . 24

3.4 Vira-wallet . . . 25

3.5 European Digital Identity Wallet . . . 25

4 Related work 29

4.1 Blockchain . . . 29

4.2 IOTA Tangle . . . 30

4.3 Zero-Knowledge Proof (ZKP) . . . 32

4.4 Self-Sovereign Identity . . . 33

5 Existing initiatives 35

5.1 Sovrin . . . 35

5.2 CredChain . . . 36

5.3 Shocard . . . 37

5.4 Casper . . . 37

6 Implementation of ValID, a digital wallet based on the SSI model 39
6.1 Wallet . . . 40
6.1.1 Onboarding . . . 43
6.1.2 GetCred . . . 49
6.2 Identity Provider . . . 56
6.2.1 CreateIdentity . . . 60
6.2.2 GetDDO . . . 61
6.2.3 CreateAP . . . 63
6.2.4 CreateUserVC . . . 64
6.3 Verifier . . . 65
6.3.1 Diagrams . . . 68

7 Installation Guide 70
7.1 Wallet_web_app . . . 70
7.1.1 Installation . . . 70
7.1.2 Usage . . . 70
7.2 Identity Provider . . . 70
7.3 Verifier . . . 71

8 Conclusion 72

Bibliography 74

Chapter 1

Introduction to Digital Identity and Self-Sovereign Identity

One of the main tasks that institutions must deal with is known as "Know Your Customer" or "KYC", which is the verification of customer credentials. Systems that manage these credentials are known by many names, one of them is "Identity Management (IDM) Systems".

A digital identity (or electronic identity, eID) is "the digital representation of a natural or legal person", and it is used to access any online service. It is composed of a set of identifiers, which are any characteristic elements that can be used to identify a person. A person or a company can be related to more than one digital identity, each of them referring to a different set of identifiers [1] [2] [3] [4].

With the arrival of the COVID-19 pandemic, digital identity has become more and more present in people's life. In Italy, for example, the number of people that started using SPID (Sistema Pubblico di Identità Digitale) increased by 100% in the first two weeks of the lockdown in March 2020. This happened because, due to the pandemic, in-person activities were forbidden, therefore many public and private organisations moved their services toward the Internet. To use these services, Italian citizens must authenticate, then users need reliable digital identities [5].

Different countries have different eID schemes, Italy for example has SPID and CIE (Carta d'Identità Elettronica). To allow citizens of different European Union countries access the services of another country with their government eIDs, the eIDAS network is used. When the authentication is performed, the needed personal attributes are transferred by the eIDAS nodes to the service providers (SPs) to make the user's authentication possible. To allow this kind of authentication, the eIDAS network must follow the eIDAS Regulation [2].

To manage the digital identity, many Identity Management (IDM) systems have been designed, but the main problem is related to the sovereignty of these identities. To face this issue, the Self-Sovereign Identity (SSI) model has been designed. In this model, the user is the owner and has full control over its private information. It is based on the use of Distributed Ledger Technologies (DLTs) to create an immutable decentralized register where information is stored. The basic architecture of an SSI model involves three actors: the user, an Identity Provider (IdP), and a Service Provider (SP). To obtain a digital identity, a user must refer to an IdP, and by using this identity he can authenticate to a SP.

In the European Union, anything that is involved in managing personal data is subject to the General Data Protection Regulation (GDPR), published in 2018. So, since DLTs are used and

due to their decentralized nature, the conformity to GDPR must be considered when designing SSI solutions. The personal information about the user is not stored in the DLT but in a Digital Wallet, a software that is under the control of the user itself, whose content is digitally signed and that the user can use to authenticate to an organisation [4], [6] [7].

Chapter 2

Study of the standards (protocols) and specifications

This chapter presents the data structures used in a SSI model. Moreover, the eIDAS regulation is described. It specifies how the eID issued from an EU country can be accepted from another EU country. Then, since SSI deals with personal information about the users, its compliance with GDPR is analysed. In the last part of this chapter is analysed SAML, the protocol used for message exchange in eIDAS.

2.1 W3C specifications

2.1.1 Decentralized Identifier

Decentralized identifiers (DIDs) are globally unique identifiers that permit verifiable and decentralized digital identity. They are independent of centralized registries, IdP, and certificate authorities: the controller of a DID can use it to prove his identity without referring to them. A DID is made of the following parts:

- Scheme identifier: the "did"prefix;
- DID Method: the mechanism to create, resolve, update, or deactivate a DID;
- DID Method-Specific Identifier.

Figure 2.1. DID structure: Scheme identifier, DID Method, DID Method-Specific Identifier [8].

Moreover, a DID can be extended to include a URI to locate a resource, and in this case, it is called DID URI. Each DID is related to a DID Subject, but a subject can have multiple DIDs to

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/ed25519-2020/v1"
  ]
  "id": "did:example:123456789abcdefghi",
  "authentication": [{
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "Ed25519VerificationKey2020",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyMultibase": "zH3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
  }]
}
```

Figure 2.2. Example of a DID Document [8].

separate different contexts and have different personas. The DID is resolved to a DID Document, which contains information related to the DID as ways to cryptographically authenticate a DID controller. A DID controller is an entity that is able to change a DID document, it might be the DID Subject. This ability is typically provided by the possession of a set of cryptographic keys. A DID can have more than one DID Controller. To be resolved in DID Documents, DIDs must be stored in a so-called "Verifiable Data Registry", which can be a distributed ledger, a decentralised file system, or any other kind of trusted data storage [8].

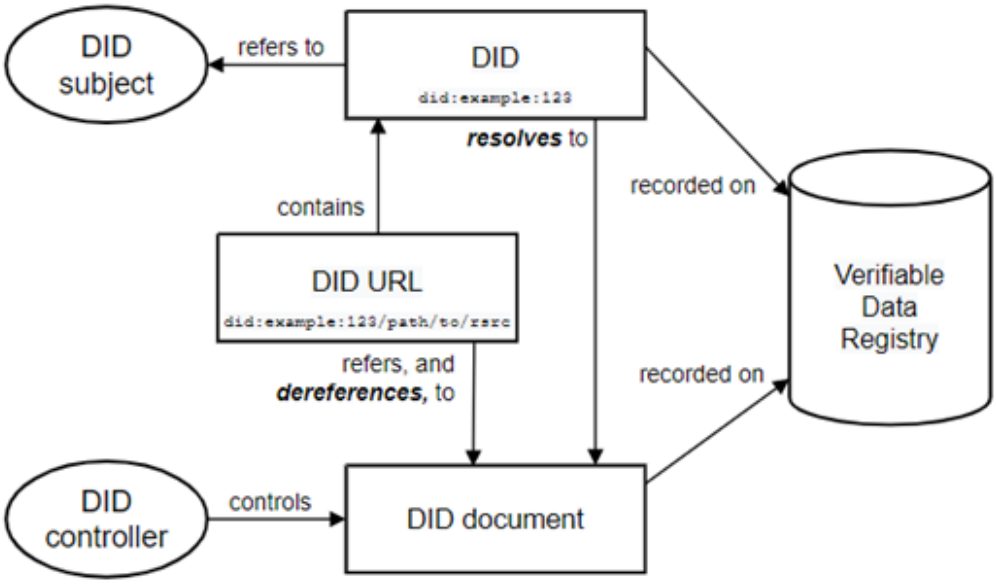

Figure 2.3. Description of a DID and related actors [8].

2.1.2 Verifiable Credential

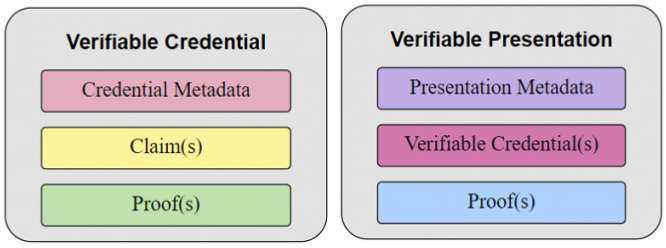

Figure 2.4. Structure of a Verifiable Credential (VC) and of a Verifiable Presentation [8].

A Verifiable Credential (VC) is the digital representation of a physical credential (e.g., a passport or a driving license). Using some technologies like digital signature, this kind of credential can be more trustworthy and tamper-evident than the physical ones. A VC is issued by an issuer by asserting claims about one or more subjects. This VC is given to the holder, that can generate from it and other VCs one or more Verifiable Presentations. Most of the times, the holder is the subject of the VC, but there are cases in which these two actors are different (e.g., a parent can be the holder of a credential whose subject is his son). The Verifiable Credential or a Verifiable Presentation generated from it is presented to a verifier, that will check it against a Verifiable Data Registry, a place where schemas and identifiers are maintained.

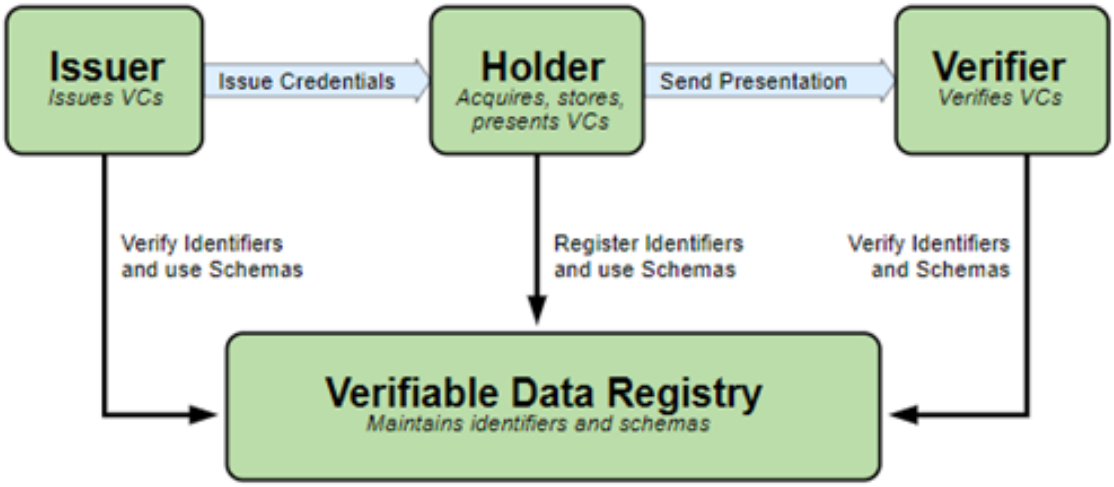

Figure 2.5. Actors involved in the VC lifecycle [8].

A Verifiable Credential is composed of a set of claims made by the same entity, an identifier, metadata describing its properties (e.g., the issuer and the revocation mechanism), and a set of proofs to detect tampering and verify the authorship of the credential.

One or more VCs can be packed in a Verifiable Presentation, that contains also metadata and a set of proofs. An example of a VC is reported in Fig. 2.6, where it is possible to identify the metadata of the credential, specifying its id, type, issuer, and issuanceDate, the claim, that

specifies the id of the subject and the assertion constituting the claim, and the proof that makes the verifiable credential tamper-evident, which is a digital signature with a certain suite and a certain verification method. An example of a Verifiable Presentation is reported in Fig. 2.7. Here it is possible to identify the Verifiable Presentation metadata, the same credential as the previous example (Fig. 2.6), and a digital signature containing the date and time on which the presentation has been created to protect it against replay attacks [8].

```
{

  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://www.w3.org/2018/credentials/examples/v1"
  ],

  "id": "http://example.edu/credentials/1872",

  "type": ["VerifiableCredential", "AlumniCredential"],

  "issuer": "https://example.edu/issuers/565049",

  "issuanceDate": "2010-01-01T19:23:24Z",

  "credentialSubject": {

    "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",

    "alumniOf": {
      "id": "did:example:c276e12ec21ebfeb1f712ebc6f1",
      "name": [{
        "value": "Example University",
        "lang": "en"
      }, {
        "value": "Exemple d'Universite",
        "lang": "fr"
      }]
    }
  },


  "proof": {

    "type": "RsaSignature2018",

    "created": "2017-06-18T21:19:10Z",

    "proofPurpose": "assertionMethod",

    "verificationMethod": "https://example.edu/issuers/565049#key-1",

    "jws": "eyJhbGciOiJSUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19..TwLtj
      PAYuNzVBAh4vGHSrQyHUdBBPM"
  }
}
```


Figure 2.6. Example of a VC [8].

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://www.w3.org/2018/credentials/examples/v1"
  ],
  "type": "VerifiablePresentation",
  "verifiableCredential": [{
    "@context": [
      "https://www.w3.org/2018/credentials/v1",
      "https://www.w3.org/2018/credentials/examples/v1"
    ],
    "id": "http://example.edu/credentials/1872",
    "type": ["VerifiableCredential", "AlumniCredential"],
    "issuer": "https://example.edu/issuers/565049",
    "issuanceDate": "2010-01-01T19:23:24Z",
    "credentialSubject": {
      "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
      "alumniOf": {
        "id": "did:example:c276e12ec21ebfeb1f712ebc6f1",
        "name": [{
          "value": "Example University",
          "lang": "en"
        }, {
          "value": "Exemple d'Universite",
          "lang": "fr"
        }]
      }
    },
    "proof": {
      "type": "RsaSignature2018",
      "created": "2017-06-18T21:19:10Z",
      "proofPurpose": "assertionMethod",
      "verificationMethod": "https://example.edu/issuers/565049#key-1",
      "jws": "eyJhbGciOiJSUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19..dBBPM"
    }
  }],
  "proof": {
    "type": "RsaSignature2018",
    "created": "2018-09-14T21:19:10Z",
    "proofPurpose": "authentication",
    "verificationMethod": "did:example:ebfeb1f712ebc6f1c276e12ec21#keys-1",

    "challenge": "1f44d55f-f161-4938-a659-f8026467f126",
    "domain": "4jt78h47fh47",
    "jws": "eyJhbGciOiJSUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19..PAnKb78"
  }
}
```


Figure 2.7. Example of a Verifiable Presentation [8].

2.2 eIDAS

With the eID it is possible to unambiguously identify a person allowing him to access the services he needs. In each European Union country, there is a different identity system (e.g., in Italy there is SPID), and to enable the recognition of an eID issued by an EU country from another one, the eIDAS Regulation has been defined and it has become effective in September 2018. With this Regulation, EU citizens can securely access online services in another EU country, where eIDs are available. Regarding privacy aspects, this regulation obeys the OECD (Organization for Economic Cooperation and Development) recommendations for the collection of users' data, achieving user data sovereignty [9] [10] [11].

Many countries of the European Union have configured and tested their eIDAS-Nodes, and to each node notified accredited Identity Providers (IdPs) supporting the country's eID schemes are linked. The EU countries can send their eID schemes to the European Commission to add them to a list of notified eID schemes, but this is not mandatory. If an EU country notifies of an eID scheme, the other countries must accept their eIDs, otherwise they are not obliged to do so. Since the different adopted solutions to identify citizens are not equivalent, in the eIDAS Regulation three Levels of Assurance (LoA) have been defined. These levels are low, substantial, and high, and they represent how much the result of the authentication can be trusted. This is related not only to the cryptographic strength of the authentication technique but also to the strength of the authentication process. This LoA has an impact on the behaviour of the SPs: they are obliged to accept eIDs with a substantial or high LoA and they are free to recognize or not electronic identification means with a 'low' level. This applies only in public sectors, but since the specification is based on SAML 2.0 also private services can comply with it [2] [9] [10] [11].

Cross-border authentication is obtained by defining the interfaces between the eIDAS-Nodes, in particular between the eIDAS Connectors and the relying parties and between the eIDAS-Services and the eID-scheme. The eIDAS Connectors are nodes requesting cross-border authentication, while the eIDAS-Services are nodes providing it. Their implementation is made by Sending and Receiving Member States (MS, the involved EU countries): the firsts send the authenticated eID, while the seconds request the authentication of the users.

Sending MS can choose between two implementations for the eIDAS-Service:

- Proxy-based: the sending MS manages an eIDAS-Proxy-Service that forwards authentication requests made by an eIDAS-Connector and authentication assertions made by the eID scheme;
- Middleware-based: the Sending MS provides a middleware that is installed on the Receiving MS. Each Receiving MS can have one or more eIDAS-Connectors: in the first case, it is called "Centralized MS", while in the second one it is called "Decentralized MS" [11].

In general, the involved actors can be divided into two groups:

- entities that set up the eIDAS Network;
- entities that use the eIDAS eID for authentication.

In the first group, it is possible to find the entities that oversee managing the eIDAS-Nodes:

- eIDAS-Node implementers and operators: entities in charge of managing an eIDAS-Node;

- Single Point of Contact: a public entity per each country that helps the Service Providers and the Identity Providers to connect to the network;
- Identity Providers: organizations that assigns to the user a secure online identity, so that he can access the online services of another EU country, provided that the IdP is connected to the eIDAS network;
- Attribute Providers: entities managing the information related to the user. To make this information available on the eIDAS network, they must connect to the eIDAS-Node of their country.

In the second group, instead, it is possible to find the following entities:

- Public Service Providers: public administrations that provide online services to EU citizens;
- Private Service Providers: private organizations can provide online services to EU citizens, even if they are not obliged to accept foreign eIDs;
- Sector-specific EU projects: cross-border projects which use eID to access some online services;
- EU citizens: they can use their national eID to access online services in other countries [11].

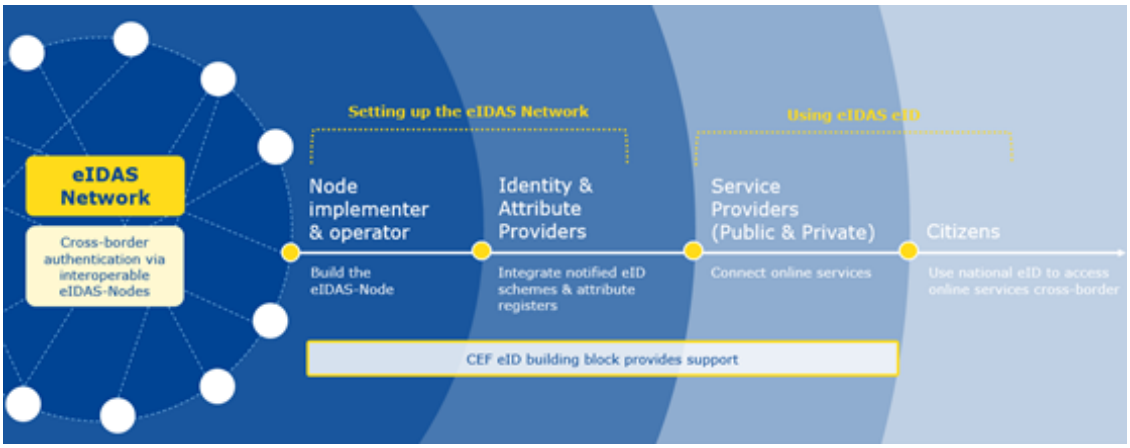

Figure 2.8. Involved actors in eIDAS Network [11].

In eIDAS, the attributes of a person are transferred for authentication purposes: four of them are mandatory (name, surname, date of birth, and eIDAS person identifier) and others are optional (e.g., current address). The set of mandatory attributes is called the eIDAS Minimum Data Set (MDS). Each country can add some additional sector-specific attributes, but then they have to decide how to get them from the Attribute Providers (APs) and how the national eIDAS-Node must process them [9] [12].

Each eIDAS-Node is composed of a Generic part and a Specific part that differs depending on the MS that implements it and on its requirements. The generic part contains the software of the European Commission, which contains several software models like the eIDAS Connector, the eIDAS Proxy Service, or the SAML engine, which is used to process the eIDAS messages since the eIDAS communication protocol is based on that standard. Here it is possible to find other two

modules, the demo IdP and the demo SP, that must be replaced by the MS that uses the eIDAS-Node with some components that can communicate with the eIDAS-Node using the Specific part. Considering Italy, the two components are replaced by the SP Proxy and the IdP Proxy, which are used to allow communication between the eIDAS-Node and the national SPs/IdPs.

In Fig. 2.9 all the components in the eIDAS-Node are shown. The eIDAS Connector exchanges eIDAS authentication messages with the eIDAS Proxy Services of other MS. Nationally, it communicates with the National SP exchanging authentication messages by using either the eIDAS protocol or the MS Specific protocol. As well as communicating with the eIDAS Connector, the eIDAS Proxy Service exchanges authentication messages with the National IdPs by using the MS Specific Part.

In the MS Specific Part, there are two components, the MS Specific Connector and the MS Specific Proxy, which process and, if needed, translate messages from the national eID scheme format to the one used by the eIDAS Connector and the eIDAS Proxy Service.

Some countries substitute the eIDAS Proxy Service with a Middleware, which is installed and run in other MS countries [9] [13] [14].

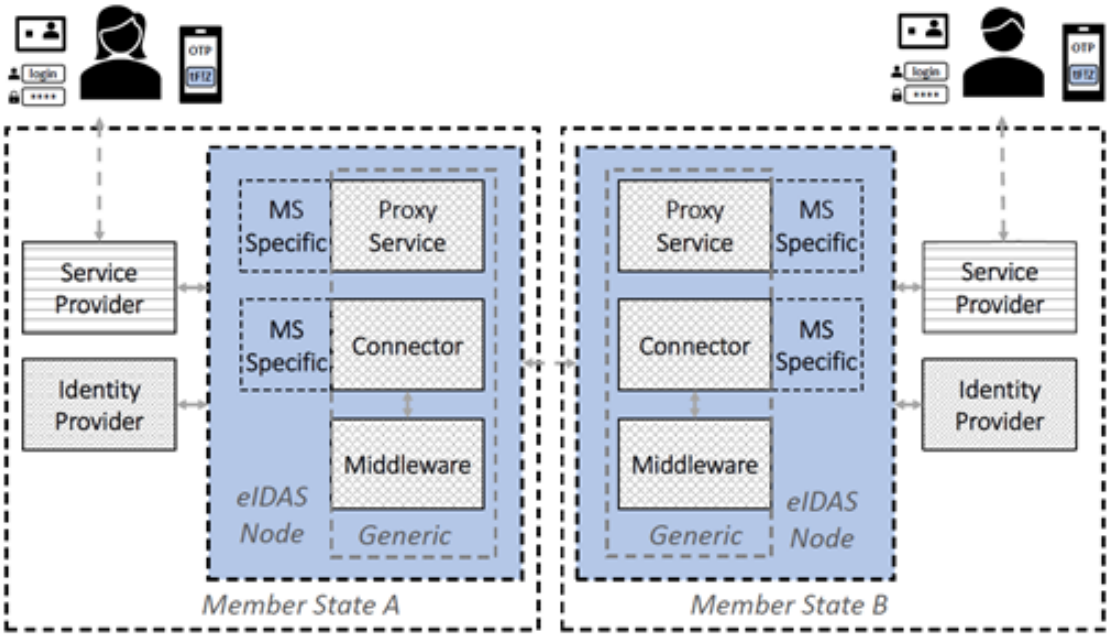

Figure 2.9. Involved actors in eIDAS Network [11].

The eIDAS Connector receives an authentication request (eIDAS-Auth-Req) from the national SP and sends it to the eIDAS Service of the Sending MS, which processes and converts the request into the local format before forwarding it to the national IdP. After that the authentication with the recognized eID is successful, the IdP sends an authentication response in local format to the eIDAS Service, which creates an eIDAS authentication response (eIDAS-Auth-Res) and sends it to the eIDAS Connector. This response contains the list of requested attributes. Then, the eIDAS Connector returns this response to the SP, and it can do it either by sending directly the eIDAS-Auth-Res, if the SP understands the eIDAS protocol, or by sending it converted into a protocol understandable by the SP.

To identify in a secure way the eIDAS Nodes an eIDAS metadata exchange is used, containing an X.509 certificate to verify the signature on the eIDAS authentication requests and responses and to decrypt the attributes contained in the response [2].

2.2.1 eIDAS Revision

With the eIDAS Regulation Revision in June 2021, the European Community proposed the European Digital Identity Framework (or EUDI Wallet). This should complement the existing notified eIDs and obligations to its acceptance for the authentication are extended also to the so-called "gatekeepers" ("big players" like Google or Facebook) and certain private sectors (e.g., banks). Being not mandatory, if citizens would use the EUDI Wallet, they would be able to prove their identity, share electronic documents, and access online services. According to the Revision, citizens would be able to link their digital identities with proof of personal attributes using digital wallets given by public authorities or by private entities recognized by a Member State [11].

Currently, 14% of the public service providers across all MSs allow cross-border authentication with eID, and the goal of this revision would be to improve this percentage [15], [11].

The main differences with the current eIDAS Regulation are:

- The MSs use central components, which are the eIDAS Nodes, while the EUDI Wallet can directly communicate with the RP;
- The EUDI Wallet can be used both online and offline (in presence), unlike other already used wallets;
- The MSs notify their eIDs having a certain level of freedom concerning the protocols and the implementation, while the EUDI Wallet needs a harmonization among the protocols due to the cross-border usage, taking into account the investments that each MS has done in their wallets;
- With the EUDI Framework, citizens can use any Wallet.

From September 2023 every EU MS must make a EUDI Wallet available for every citizen who wants to use it [15]. To make the European Digital Identity a reality as soon as possible, the EC proposal recommended the Member States set up a toolbox since September 2022 containing the technical and legal aspects [11]. Under the new Regulation, Member States will offer citizens and businesses digital wallets allowing them to link their national digital identities with proof of other personal attributes (e.g., driving license, diplomas, bank account) [11].

2.3 General Data Protection Regulation (GDPR)

The General Data Protection Regulation (GDPR) has been approved in 2016 by the European Parliament and became effective in 2018. It regards the processing of the personal information of every alive EU citizen outside the personal sphere by individuals, companies, or organisations [16], [11].

The regulation distinguishes between controllers, processors, and data subjects, and it enforces obligations to the first two entities. It applies to personal data, which means any information linked to an identifiable natural person (Art. 4.1 GDPR).

Since the SSI manages personal information of the users, it should be compliant with the GDPR rules. In this scenario, there are four kinds of data: DIDs, credentials, revocation of credentials, and hashes of the first three elements. Credentials and revocations are usually considered personal data, while for DIDs and hashes other considerations are needed. DIDs do not allow the data subject identification, but their repeated usage could permit it: when used more than once, DIDs allow linking credentials or attributes. For what regards hashes, a cryptographic hash function is a non-reversible one-way function, so it is not possible to recover the original data from its hash. However, in some cases, the original information could be obtained by performing brute force on the hashed value. To consider the hash as personal data, it must originate from personal information [17].

In Article 5 of the GDPR regulation, seven key principles have been discussed:

- Lawfulness, Fairness, and Transparency: the processing of personal information must be fair;
- Purpose Limitation: data must be collected only for a precise and explicit scope;
- Data Minimization: the data collected should be only those necessary;
- Accuracy: data must be analyzed to avoid having incorrect or misleading information;
- Storage Limitation: data should be collected only for the needed time;
- Integrity and Confidentiality: GDPR does not specify how these two properties should be achieved but they must be considered;
- Accountability: the roles involved in processing personal information must have the responsibility to respect all GDPR principles.

The compatibility between SSI and GDPR must be analysed considering each specific SSI solution, but some general observations can be done:

- Lawfulness, Fairness, and Transparency: the holder owns his personal data and every action on it is done after his consent. However, there are big challenges for an SSI solution to be GDPR-compliant. For example, it is not clear when personal information is properly anonymised;
- Purpose Limitation: the holder can decide with whom and for which purpose to share his personal data giving his consent, and this consent can be stored on the blockchain. However, the purpose declaration depends upon the SSI solution analysed;
- Data Minimization: in most SSI solutions, personal information is stored off-chain, but transactions are stored on the blockchain, an append-only database replicated in different locations, and these aspects do not comply with GDPR regulation;
- Accuracy: since personal information is stored on the holder's device, it is easy to erase and update it, and this follows the Right to be Forgotten and the Right to Erasure of the GDPR. However, DIDs and other information are stored on the blockchain, and this can be a challenge for these two rights;
- Storage Limitation: erasing information from the blockchain may be difficult;
- Integrity and Confidentiality: most SSI solutions comply with this principle using techniques as cryptographic solutions;

- Accountability: SSI systems based on permissioned blockchains can achieve greater accountability where a competent authority can perform actions that aim for this. However, for the solutions that use a permission-less blockchain, it is difficult to respect this principle.

In conclusion, GDPR adapts better to a centralized network, and this goes in contrast with the use of the blockchains from the SSI solutions, even if those based on a permissioned blockchain are able to follow most of the GDPR principles [3].

2.4 SAML 2.0

The Security Assertion Markup Language (SAML) is a standard that defines an XML-based framework for the exchange of security information developed by OASIS (Organization for the Advancement of Structured Information Standards). There have been defined three versions of SAML, the currently adopted is 2.0.

The base object of SAML is the Assertion, an XML document that contains a statement about the considered subject. SAML defines three possible statements:

- Authentication statement: created by the party that authenticates the users after a successful login;
- Attribute statement: used to provide attributes of the subject;
- Authorization decision statement: used to define an action that the subject of the assertion can perform.

Among the reason behind the usage of SAML standard, the most important are:

- It provides Single Sign-On facility;
- It can be used to achieve federated identity;
- It is a modular framework that can be used outside of a native SAML-based protocol context.

Assertions are created by an asserting party usually after a Request from a relying party. The request must be created for a specific type of assertion, that will be contained in a SAML container and put in a Response.

SAML Assertions are transferred using a certain protocol, and the SAML bindings define how this protocol is used. Assertions, protocols, and bindings can be combined to support specific use cases producing SAML Profiles.

The following are two important aspects of the SAML environment:

- Metadata: used to share configuration information between SAML parties;
- Authentication Context: used to provide information about the type and the strength of the authentication performed by an Identity Provider [18].

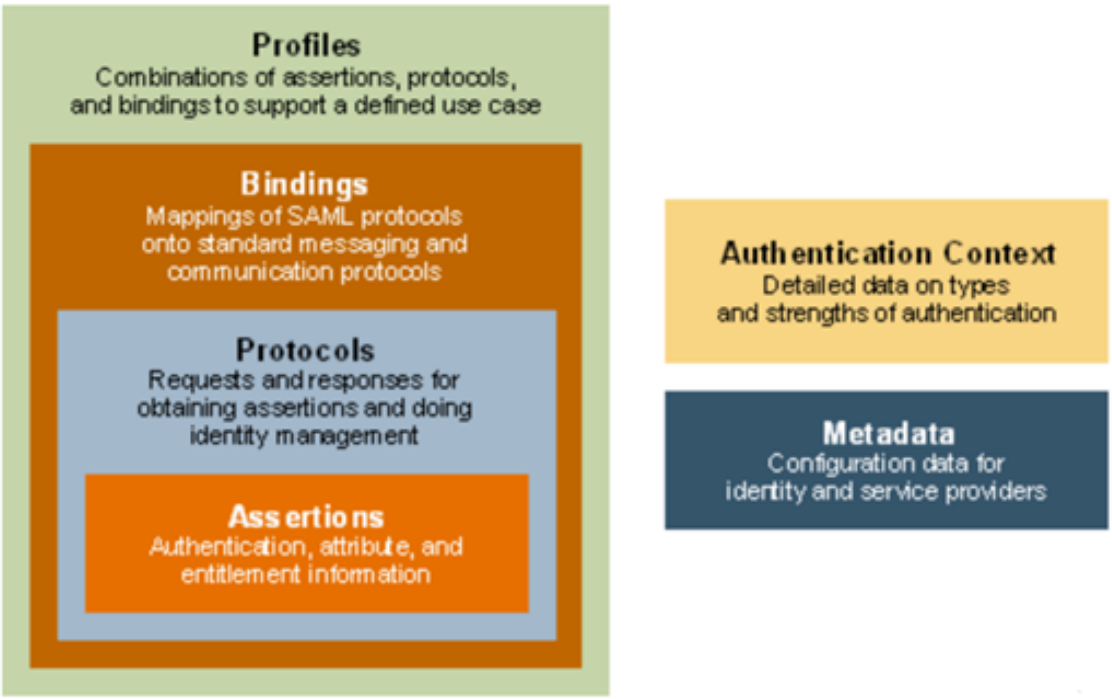

Figure 2.10. SAML elements [18].

```
<saml:Assertion
    MajorVersion="1" MinorVersion="0"
    AssertionID="192.168.1.1.12345678"
    Issuer="Politecnico di Torino"
    IssueInstant="2007-12-03T10:00:00Z">
    <saml:Conditions
        notBefore="2007-12-03T10:00:00Z"
        notAfter="2007-12-03T10:05:00Z" />
    <saml:AuthenticationStatement
        AuthenticationMethod="password"
        AuthenticationInstant="2007-12-03T10:02:00Z">
        <saml:Subject>
            <saml:NameIdentifier
                SecurityDomain="polito.it" Name="alioy" />
        </saml:Subject>
    </saml:AuthenticationStatement>
</saml:Assertion>
```

Figure 2.11. Example of a SAML Authentication Assertion.

```
<saml:Assertion
    MajorVersion="1" MinorVersion="0"
    AssertionID="192.168.1.1.12345678"
    Issuer="Politecnico di Torino"
    IssueInstant="2007-12-03T10:00:00Z">
    <saml:Conditions
        notBefore="2007-12-03T10:00:00Z"
        notAfter="2007-12-03T10:05:00Z" />
    <saml:AttributeStatement>
        <saml:Subject>
            <saml:NameIdentifier
                SecurityDomain="polito.it" Name="alioy" />
        </saml:Subject>
        <saml:Attribute
            AttributeName="Dipartimento"
            AttributeNameSpace="http://polito.it">
            <saml:AttributeValue>
                DAUIN
            </saml:AttributeValue>
        </saml:Attribute>
    </saml:AttributeStatement>
</saml:Assertion>
```

Figure 2.12. Example of a SAML Attribute Assertion.

Fig. 2.11 reports an example of an authentication assertion, which contains the following elements:

- "saml:Assertion": it indicates that the element is an Assertion. Inside this tag are specified the major and minor versions, the assertion ID, the issuer, and when the assertion has been issued;
- "saml: Conditions": it specifies the conditions under which the assertion is valid. If the conditions are not understood by the receiver, the assertion must be rejected;
- "saml:AuthenticationStatement": it contains the result of the authentication and explains the authentication method, when the authentication has been performed, and the subject that has been authenticated. To identify the subject is necessary to specify the name identifier and the security domain to which it belongs.

The "saml:AttributeStatement" tag in Fig. 2.12 indicates that this is an attribute assertion. Inside it, the subject is specified as in the authentication assertion. The "saml:Attribute" tag is used to define the attribute by specifying its name, its namespace, and its value.

The "saml:AuthorizationStatement" tag in Fig. 2.13 explains that it is an Authorization decision assertion, and it specifies the decision, the resource that needs to be accessed, and the subject of the decision. Often the assertions are used together in a schema called the "Producer-Consumer model", in which there are several entities that at the same time produce and consume (receive) assertions.

```
<saml:Assertion
    MajorVersion="1" MinorVersion="0"
    AssertionID="192.168.1.1.12345678"
    Issuer="Politecnico di Torino"
    IssueInstant="2007-12-03T10:00:00Z">
    <saml:Conditions
        notBefore="2007-12-03T10:00:00Z"
        notAfter="2007-12-03T10:05:00Z" />
    <saml:AuthorizationStatement
        Decision="Permit"
        Resource="http://did.polito.it/m2170.php">
        <saml:Subject>
            <saml:NameIdentifier
                SecurityDomain="polito.it" Name="alioy" />
        </saml:Subject>
    </saml:AuthorizationStatement>
</saml:Assertion>
```

Figure 2.13. Example of a SAML Authorization Decision Assertion.

```
<samlp:Request
    MajorVersion="1" MinorVersion="0"
    RequestID="128.14.234.20.12345678">
    <samlp:AuthenticationQuery>
        <saml:Subject>
            <saml:NameIdentifier
                SecurityDomain="polito.it" Name="alioy" />
        </saml:Subject>
    <samlp:AythenticationQuery>
</samlp:Request>
```

Figure 2.14. Example of a SAML Authentication Request.

In “samlp” in Fig. 2. 14, “p” stands for protocol. The “saml:AuthenticationQuery” specifies that it is an authentication assertion request, and inside it, the Subject of the request is described. The entity that receives an assertion must trust the one that emits it. This trust is established by means of a secure channel with mutual authentication or with the authentication of at least the assessing party. Another possibility, instead of using a secure channel, is to use XMLsignature over the SAML object with a shared or public key.

Chapter 3

What is a digital wallet?

A digital wallet is the electronic representation of a physical wallet. As the physical wallet contains physical credentials like an identity card or a driving license, the digital wallet contains digitally signed verifiable credentials used by the wallet holder to demonstrate his identity and some attributes related to it [7].

A digital wallet is a software that enables the storage, management, and sharing of data related to the identity of the holder and provides secure storage for cryptographic material associated with this data. Using a wallet, the holder can have full control over his identity, which means he has the ability to delete some information from it and he can decide which kind of personal information to share and with whom [19].

In the following part of this section, some of the existing digital wallets are analysed.

3.1 Connect.me

Connect.me is an open-source self-sovereign digital identity wallet created by Evernym, a platform for verifiable credentials ideated by the creators of Hyperledger Indy and Sovrin, which is based on the Sovrin Network, an instance of Hyperledger Aries [20].

Sovrin ecosystem allows to create private and secure connections with other entities in the ecosystem, manage digital credentials and present digital proofs. It uses Hyperledger Ursa to perform cryptographic operations, its implementation is based on React Native, and it can be executed both on iOS and Android [21].

Since it is based on the Sovrin network, it uses W3C standards like DID and VC. In this network, DIDs are resolved using DIF Universal Resolver, and VCs are managed with the ZKP standard. Personal information about the user is stored on the Connect.Me wallet, which is accessed only by the Sovrin Agents, while on the Sovrin network DIDs, schema definitions, credential definitions, and revocation registries are stored [22] [23].

3.2 Jolocom

It is an open-source protocol designed to operate on public blockchain infrastructures like the Ethereum one to identify entities using DIDs. These are stored on the blockchain, while personal information is stored off-chain under the sovereignty of the user [24].

Jolocom uses standards largely adopted in the SSI community as DID and VC, and its goal is to simplify the management of digital identities. In this context, an identity is a combination of a DID and a set of keys, whose number depends upon the requirements of the DID method used [25].

To create, issue, manage and verify verifiable credentials, the Jolocom Agent is used. The user can control his personal information and prove his identity using the Jolocom SmartWallet 2.0. This application has been developed using React Native, and it is available for both iOS and Android. The major strengths of this wallet are the following:

- It is compliant with GDPR regulations;
- It has been designed considering the user experience, and this leads to an intuitive interface;
- It is open source on GitHub;
- The Jolocom technology is agnostic with respect to the network, improving users' flexibility and freedom [26].

3.3 Trinsic

Trinsic is an identity wallet and verifiable credential infrastructure based on W3C standards and Hyperledger Aries [27].

The Trinsic platform is composed by the following elements:

- Trinsic Core: the set of APIs that the users can use for demonstrating things about themselves. These APIs can be used to issue, share, and verify credentials. It has been updated to version 2 increasing scalability, interoperability, and performance;
- Trinsic Ecosystems: using the Trinsic platform it is possible to set up your trust ecosystem, deciding what credentials are exchanged between the participants, what are the allowed participants, and how they can trust each other. The ecosystem is a network of companies and consumers that must be able to share and verify data. The organisations that implement the ecosystem are called "providers", and the ecosystems give them the possibility to be interoperable with other digital trust ecosystems, even if they use different standards, via the Trinsic's Schema Registry;
- Trinsic Studio: a general-purpose dashboard to interact with all Trinsic's products. Organisations can use it to deal with the issuance, management, and verification of credentials. Moreover, the platform gives users support in case they need to revoke a credential;
- Identity Wallets: customers can use them to manage their credentials, certificates, and licenses and prove their identity using their phones. With the Trinsic Wallet SDK it is possible to use everywhere digital wallet capabilities, and since it supports all the common programming languages it is easy to integrate it into existing applications.

The platform is tech-agnostic in the sense that it supports different VC implementations, cryptographic suites, and DID methods [27].

Personal information about the user is stored in the wallet application on the user's phone, while DIDs are stored on the blockchain [27], [28].

Trinsic's pricing model is based on DIDComm usage, which is the number of communications that a user has with the others. In these pairwise connections, each peer generates a new identifier (pseudonymous) and exchanges DIDs. These DIDs act as a public key for the encryption, and the related private key is used by the related user. This protects from man-in-the-middle attacks, and since the keys are stored locally in the user's digital wallet, there is not the possibility for a third party to use them. To start a connection in Trinsic, first of all, a connection invitation must be created, and then this invitation must be sent to the party with which the user wants to connect. This is usually done using a QR code that encapsulates the invitation. Once the connection is established, the credential exchange can start. Trinsic Studio also gives the possibility to have the so-called "Connectionless Exchanges", ways to exchange and verify credentials without establishing a persistent connection first, and this is done using URLs that are encoded into a QR code.

Moreover, in the Trinsic platform a "Proactive Credential Sharing" is allowed: normally, a user shares a credential after a verification request by someone else, but in this case, the wallet holder can decide autonomously to share some information [28].

3.4 Vira-wallet

The Vira Wallet is an open-source Self-Sovereign identity wallet based on the IOTA identity protocol and built upon the IOTA mainnet. It has been developed by Tangle Labs, a working team inside IOTA whose work is focused on digital identity management. The application is written in React and it is available for both iOS and Android. Focusing on interoperability and standards, it has been decided to use W3C standards as DIDs and VCs.

Following the SSI philosophy, Vira wallet gives the user full control over his identity: personal information is stored only client side and it is possible to have a partial disclosure of the credentials. This selective disclosure is very important for the privacy of the user because he can decide to provide only the necessary information without revealing a larger amount of data, and this follows the principles of the Self-Sovereign identity.

It provides the users with functionalities such as QR codes for credentials sharing and the possibility to use NFC [29] [30].

The application is part of an identity suite that is composed of the following elements:

- Vira Wallet;
- Certify: an application that allows to issue and organize digital identities and VCs;
- Cloudvault: a VC secure cloud storage solution where it is possible to store the credentials which the user does not want to store only locally on his mobile phone. VCs are stored encrypted, and their sharing is done selectively with permissioned domains [29] [31];

3.5 European Digital Identity Wallet

It is a European Commission project born in June 2021 from the eIDAS Regulation Revision that aims to build a unified digital identification system in Europe. By September 2023, every EU MS must provide a EUDI Wallet to every citizen who wants to link their national digital identities with proof of other personal attributes [32].

The roles that will be involved in this project are the following:

- End users of EUDI Wallets: natural or legal persons using it to manage attestations and particular attributes about themselves.
- EUDI Wallet Issuers: MSs or organizations recognized by MSs.
- Providers of Person Identification Data (PID): they might be, e.g., the same organisations that today provide official identity documents, EUDI Wallet issuers, and so on. They must verify the identity of the user, provide him PID securely, and provide information useful for the relying party to verify the validity of the PID.
- Providers of Registries of Trusted Sources: they need to provide a registration service for the entities whose identity needs to be verified in a trustworthy way, maintain a registry, and enable third parties to access the registry.
- Qualified Electronic Attestation of Attributes Provider: Qualified EAA would be provided by Qualified Trust Service Providers (QTSPs). They must maintain an interface for requesting and providing QEAA, including an interface for mutual authentication with EUDI Wallets and another one for the verification of the attributes.
- Non-Qualified Electronic Attestation of Attributes Providers: Non-qualified EAA can be provided by any trust service provider. They must provide a way to ask for and obtain EEA, and they need to be compliant with EUDI Wallet specifications.
- Qualified and Non-Qualified Certificates for electronic signature/seal providers: the possibility for the user to sign by means of a qualified electronic signature or seal is a requirement from the European Parliament. The possibility to sign by means of a non-qualified signature or seal is instead something that the wallet may provide.
- Provider of other Trust Services: "other trust services" means something like timestamps, and the specifications for this role still need to be discussed.
- Authentic Sources: public or private repositories or systems recognized by relying parties containing attributes of a natural or legal person. They may need to maintain user authorization to give QEAA providers access to their personal information.
- Relying Parties: natural or legal persons that rely upon an electronic identification or a trust service. To rely on the EUDI Wallet, they have to notice it to the MS, and they need to maintain an interface with the EUDI Wallet to ask for attestations with mutual authentication. When they receive an attestation, they must authenticate it.
- Conformity Assessment Bodies (CAB): entities nominated by the MSs that certify the EUDI Wallets and audit regularly the QTSPs.
- Supervisory Bodies: entities that MSs must notify to the Commission and that supervise QTSPs and take actions if needed regarding non-qualified TSPs.
- Device Manufacturers and related entities: for a set of purposes, the EUDI wallet has a set of interfaces with the devices they use. The proposal imposes some limitations with respect to the devices or the services that can interact with the EUDI Wallet.
- Catalogue of attributes and schemes for the attestation of attributes providers: it contains technical specifications and standards.

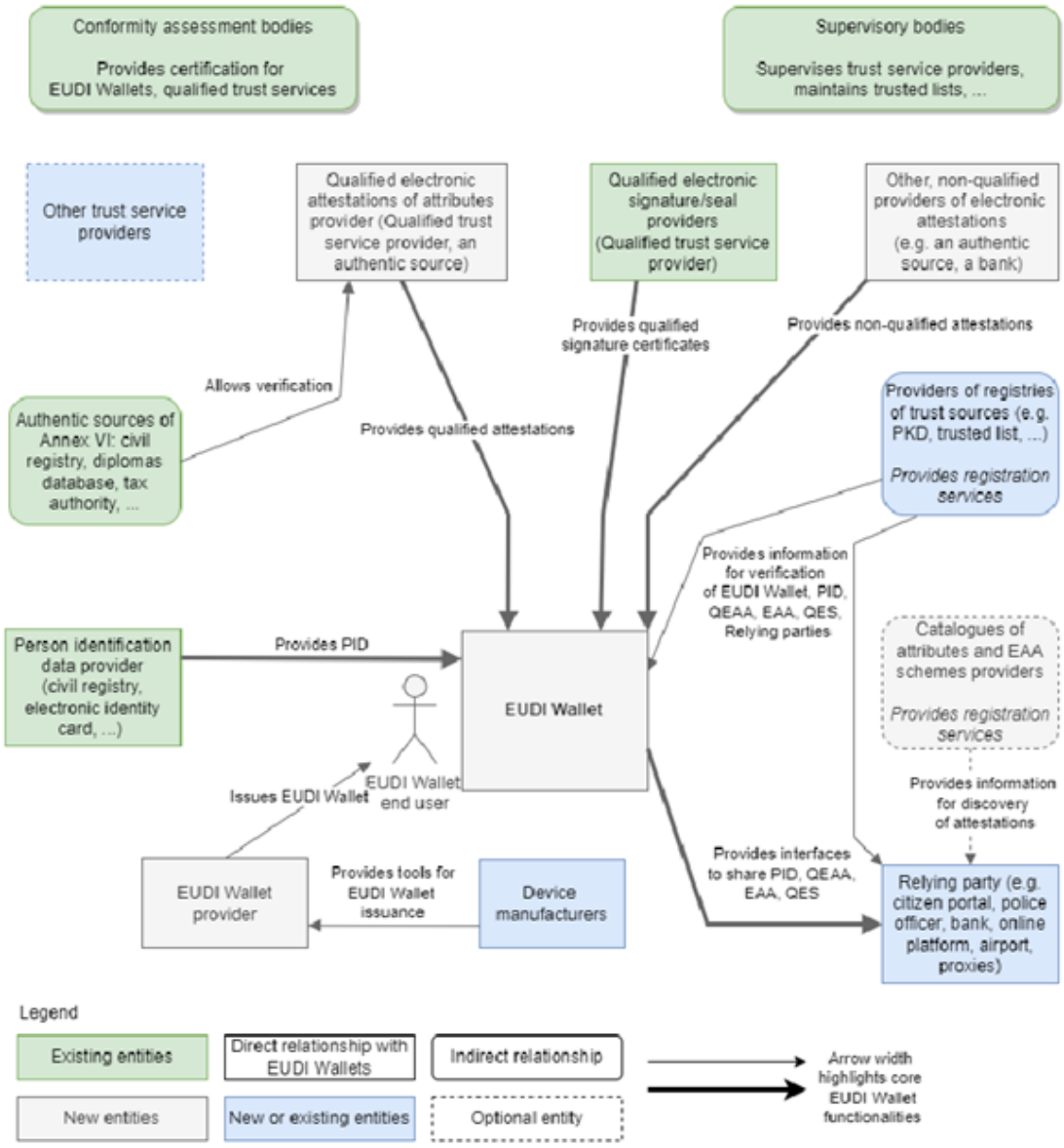

Figure 3.1. EUDI Wallet roles [33].

For what regards the functionalities, the wallet should:

- Perform electronic identification, store, and manage QEAA and EAA.
- Manage attestations made by providers, QEAA, and EAA.
- Manage cryptographic functions.
- Perform mutual authentication between external entities and the EUDI Wallet.
- Manage and share with RPs PID, QEAA, and EAA.

- Support user awareness and make explicit the authorization mechanism.
- Perform signatures using qualified electronic signature/seal (QES).
- Provide interfaces to external parties.

Those features are represented in Fig. 3.2 [33].

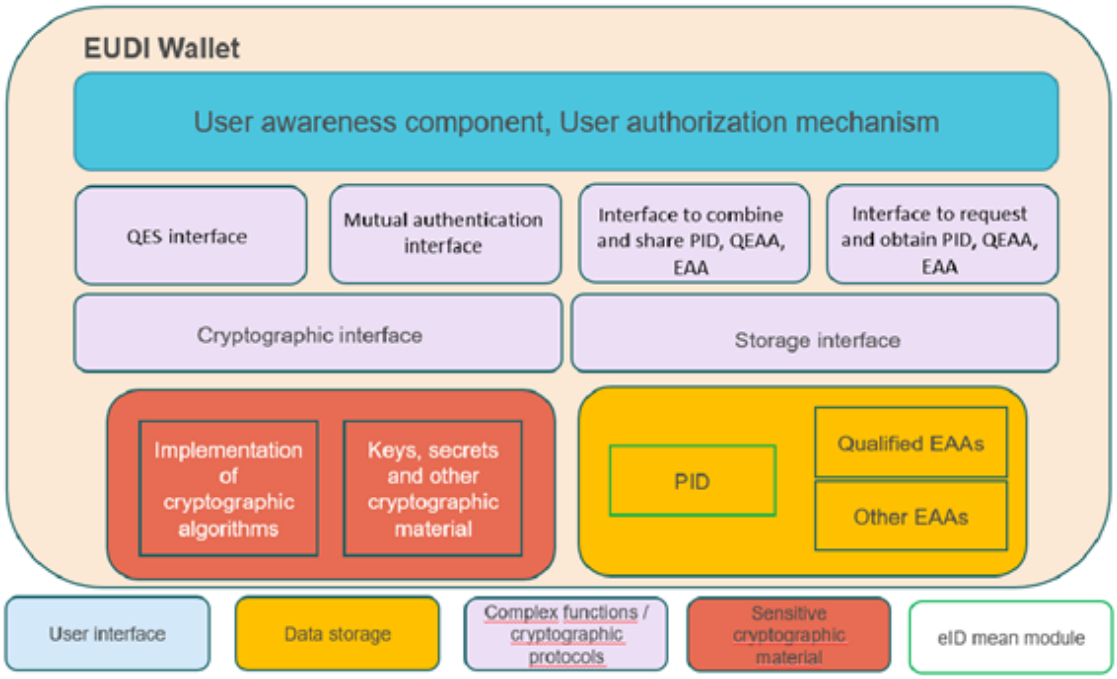

Figure 3.2. EUDI Wallet components [33].

Chapter 4

Related work

This chapter underlines the differences and characteristics of two kinds of DLTs, blockchain and IOTA Tangle. Then, since in some SSI solutions Zero-Knowledge Proofs are used to share personal information about the user, they are described in the following paragraph. The final part is dedicated to the description of the reference model for this project, the SSI.

4.1 Blockchain

A Distributed Ledger Technology (DLT) is a technology that enables the management and the update of a kind of digital data structure that is distributed through multiple computers. With DLTs more entities can maintain a copy of the database and can contribute to it, instead of having a single entity that possesses the database.

A blockchain is a peer-to-peer (P2P) DLT, in which a set of transactions is stored in a chain of blocks. It is an immutable read-only data structure and systems based on this technology work in a distributed way, with a set of nodes that communicate with each other through P2P communications. These nodes are devices that support the network by keeping a copy of the ledger or a subset of it. Nodes can perform transactions, operations that cause the exchange or the transfer of digital assets; if they perform a transaction, they are considered blockchain users. The blocks can contain any number of transactions, up to a certain block size. To update the blockchain, it is needed to reach the so-called consensus, which is related to the correct state of data on the system, because all nodes must share the same data. To achieve this, a consensus protocol is used, and it is different depending on the implementation of the blockchain. For example, the Bitcoin blockchain uses a Proof-of-Work (PoW) based consensus mechanism. In general, mechanisms can belong to two main classes: Proof-of-X based algorithms and Byzantine Fault Tolerant algorithms. An important aspect is constituted by the smart contracts, programs that perform some actions when some conditions are satisfied in the system.

This kind of technology aims at providing users with some benefits like integrity, authenticity, security, and non-repudiation in a distributed environment. Moreover, it offers immutability, since it is not possible to modify transactions once they have been published, and pseudonymity to its users.

The blockchains can be permissionless or permissioned. In the first case, anybody can participate in the network and in the consensus process. In the second one, users have restrictions either

on writing or on reading and writing. In the first scenario, any modification on the blockchain can happen only if done by a restricted group of users. In the second one, the so-called full-permissioned blockchain, the participants are decided in advance, and they are the only ones that can interact with the blockchain. In permissionless blockchains users are pseudonymous, but a malicious node can still perform actions on the network. In a full-permissioned blockchain, instead, security risks are reduced since transactions are not visible to everyone. Permissioned blockchains can be subdivided into private blockchains and consortium blockchains, depending on where the participants are (if inside the same organisation or in a consortium). In a permissioned blockchain, only some users can validate transactions. In this way, significant scalability improvements can be achieved.

Bitcoin uses a permissionless blockchain, and it is built to avoid double spending and Sybil attacks. This blockchain has some limitations, for example, there is no complete anonymity and blocks have a limited size. This means that the number of validated transactions per second is low. Moreover, to validate blocks the so-called miners must perform computationally heavy operations, so the eco-sustainability of the validation process is another issue [34].

Another permissionless blockchain is Ethereum, whose peculiarity is the introduction of smart contracts. If smart contracts invocation can cause a change in the ledger, it always happens in the context of an atomic transaction. If this transaction is valid, then, it is inserted in a block, that is broadcasted throughout the network so that all peers can execute it. Being permissionless, also this blockchain has low performance and confidentiality issues (data is publicly accessible as it is replicated in all nodes). An example of a permissioned blockchain is instead the Hyperladger Fabric. Since the participation in the consensus protocol is limited, in this blockchain it is possible to use the Byzantine Fault Tolerant (BFT) protocols, which have better performances in terms of confidentiality. The main disadvantages of using this kind of blockchain are the reduction of decentralization and the predefined set of nodes performing transaction validation [35].

4.2 IOTA Tangle

In 2016, the IOTA Foundation launched the first distributed ledger based on a directed acyclic graph (DAG) called “the Tangle”. The goals of the Tangle are to be highly scalable, to apply no fees (it does not require miners), and to perform near-instant transfers [36][37].

IOTA Tangle is not a traditional blockchain, it works using a Tangle that stores transactions history by linking parent and child transactions, providing a single source of truth.

IOTA architecture is composed of the following parts:

- Clients: users of the network that send transactions to nodes to include them in the Tangle.
- Nodes: devices connected between them that create the IOTA network and that ensure the integrity of the tangle.
- Tangle: the data structure replicated on the nodes [38]

In the structure represented in Fig. 4.1, the squares represent transactions, while the arrows represent references between transactions. The older transactions are on the left of the image. Each transaction can validate up to other eight ones that happened before, and each transaction can be in one of the following states:

- Confirmed (green in the image): a transaction is in this state if all the “tips” refer directly or not to it. If a transaction is green, it means that the consensus on it has been reached.

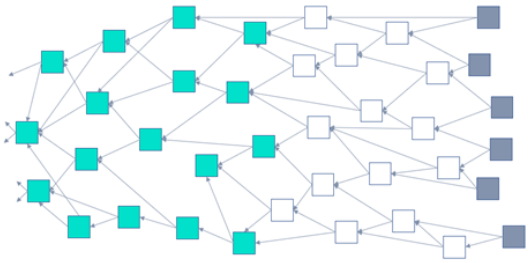

Figure 4.1. Tangle structure [39].

- Pending/unconfirmed (white in the image): a transaction is in this state if not all the tips refer to it.
- Tip (grey in the image): a transaction is in this state if it has not been referenced by any other transaction.

The lifecycle of a transaction is the following:

- Creation: the transaction must contain the recipient address, the sender, the value, and optionally a message.
- Signing: the transaction is signed with the private key of the creator.
- Tip selection: the transaction chooses up to eight tips to reference.
- Proof of Work: it is not needed for the consensus but for spam protection. It requires the resolution of a small arithmetic problem and a check of the selected tips.
- Execution: the transaction is sent to a node that will forward it to all its neighbours.

The need for a Tangle Data Structure comes from the limitations of the Blockchain one. In the Blockchain Data Structure, a block can be added only to a previous block since there is a chain of subsequential blocks, as described in Fig. 4.2.

Figure 4.2. Representation of a new block addition to a blockchain [39].

In the Tangle Data Structure, instead, a transaction can be appended anywhere and references up to other eight transactions. It is a more scalable solution [39].

Figure 4.3. Representation of a new block addition to the Tangle in two different states [39].

4.3 Zero-Knowledge Proof (ZKP)

Zero-Knowledge Proofs (ZKPs) are protocols between a prover and a verifier which allow the prover to demonstrate to the verifier that something is true without revealing anything more than that specific statement is true. In this context, a prover is a party that tries to demonstrate a claim and a verifier is a party that has to validate the claim provided by the prover [40] [41].

A Zero-Knowledge protocol must satisfy a set of conditions:

- Correctness: if the information given by the prover is true, then a ZKP method must return true;
- Soundness: if the information given by the prover is false, then a ZKP method must return false;
- Zero-knowledge: the only information revealed by the method is whether the prover tells the truth or not [40] [42].

Two kinds of ZKP algorithms exist:

- Interactive ZKPs: they include a series of mathematical challenges which have to be solved by the prover to convince the verifier that his claim is true. The main disadvantage of this scenario is that it has limited transferability, to prove the same claim with someone else the process needs to be repeated;
- Non-Interactive ZKPs: they do not involve any interaction between the transacting parties. In this scenario, a prover creates a proof that anyone can verify, and the verification can happen even in a second moment [43][44].

The blockchains are transparent, and so there is limited privacy because to validate transactions the source destination and the contents of the transactions are visible to anyone. To improve privacy and security, several blockchain protocols use ZKPs, in particular non-interactive ZKPs, because of their capability to be verified by anybody without the need to have an interaction between the prover and the verifier [45].

4.4 Self-Sovereign Identity

The Identity Management (IDM) models have developed through the years, and Self-Sovereign Identity (SSI) is referred to as "IDM 3.0" because it is the last IDM model proposed. The oldest one is the Centralized IDM model (IDM 1.0), in which users receive credentials from an organisation to access its services, with a trust relationship based on a shared secret, which in most cases is the pair username and password. This model has two major issues: the information about the identity of the user is managed by the organisation and this procedure must be repeated by the users with every organisation providing useful services for them. Moreover, a large amount of information stored in a central location is an incentive for attackers. These issues were fixed in the IDM 2.0 model, the Federated IDM model. Here, Identity Providers (IDPs) to avoid the management of personal information about the users by the organisations and Single-Sign On (SSO) facility to avoid users to manage several different credentials were introduced. However, in this model, the information about the users is managed by the IDPs, and to give full control of their information to the users the federated IDM model was improved by creating the Self-Sovereign IDM model [1] [7] [46].

SSI is a decentralized digital identity model that gives entities in the world a digital identity that allows secure, privacy-protected, trusted, and self-governed access to various services in the digital world by using verifiable credentials. Using this model, the identity owner controls his identity and the related attributes, being able to decide with whom and for which purpose sharing personal information by giving his consent. To have decentralization, DLTs are used, and this allows users to perform operations without the need for authorisation from a central authority. SSI gives more power to the users, but also the identity management process becomes very efficient and less onerous for organizations since the issues are reduced.

In an SSI ecosystem three main roles are involved:

- Issuer: a trusted entity that issues verifiable credentials to the holder;
- Holder: the owner of the identity that gets the credentials from an issuer. The credentials are stored inside its digital wallet and are used to create a verifiable presentation that is presented to the verifier during the authentication;
- Verifier: normally a service provider that asks the holder for some credentials and verifies the signatures within the presentation to check the validity of the credentials.

It is meaningful to note that the issuer is no more involved in the process after the verifiable credential is created [3] [22] [6] [47].

A distributed ledger is used to establish a trust relationship in the identity management process. Another important element of the SSI framework is the Decentralized Identifier (DID), a permanent and universally unique identifier related to an identity and linked to a set of verifiable credentials (VCs), which are tamper-evident and privacy-preserving credentials made by an issuer, whose validity can be verified using its digital signature, whose authenticity can be checked with the public DID of the issuer on the distributed ledger. Public DIDs and some other public credentials chosen by the identity owner are stored on the blockchain in the form of DID documents, while private DIDs and private data are maintained in the digital wallet of the identity owner. In this scenario, a claim is an assertion made on an entity, while a credential is a set of claims used by an entity to prove its identity. A credential is a Verifiable Credential (VC) if it is signed by the issuer that creates it or by another one that can confirm its truth [3] [22] [47].

To understand what sovereignty means, different aspects need to be considered. Cyberspace is considered without constraints, but physical cyberspace is composed of different components

that are placed in different states, and each state can exercise its sovereignty over the cyberinfrastructure inside its territory. Various factors can influence the sovereignty of an identity, for example, different states may enforce different rules over citizens' identity reducing its usability, or different identity networks can use different regulatory schemes. So, sovereignty is not a binary feature, and the main aspects that can give us an idea about its level are the sovereignty of an identity within the identity network and the sovereignty of an identity owner in the state in which he operates [47].

With SSI, the user is the only owner of his identity, he should be able to perform all the actions related to his identity or chose someone to do it, and no one should be able to use his identity or revoke it on his behalf. When the user has to provide some verifiable credential to prove his identity, he can decide to share the full credential, a claim, or Zero-Knowledge-Proofs (ZKPs) [3].

The lifecycle of an IdM based on blockchain is composed of four phases:

- Registration: the DID and the associated public keys are created on the distributed ledger;
- Authentication: the entity must prove the ownership of the DID;
- Issuance of credentials: the status of the credential is stored in the decentralized ledger with a cryptographic proof of issuance and a timestamp;
- Verification: the cryptographic proofs in the blockchain are validated [23].

Chapter 5

Existing initiatives

In this section, several SSI initiatives are described.

5.1 Sovrin

Sovrin is an open-source IDM system developed by the Sovrin Foundation. It is based on the Sovrin network, which is an instance of the Hyperledger Indy framework, an open-source public permissioned distributed ledger created specifically for decentralized identities. [24] [2]

The operations referring to the identity are managed by the Sovrin Governance Framework (SGF), developed by the Sovrin Governance Framework Working Group (SGFWG). The nodes of the network contain a copy of the ledger and are managed by the Stewards, trusted organisations that are the only ones that can operate nodes and participate in the consensus protocol. [7] [22] [46].

This solution follows the W3C standards and uses ZKPs to manage VCs and the Decentralized Identity Foundation (DIF) Universal Resolver to resolve DIDs. [24] [5]

Sovrin architecture is composed of four layers:

- Ledger Layer: the ledger stores credential definitions, DIDs, schema definitions, and revocation registries. Blocks can be added only by Stewards and validation is performed by using the Indy Plenum algorithm.
- Agent Layer: in this layer, peer-to-peer connections between two entities happen. These connections can take place by using the so-called "agents", programs required by any entity in the network that wants to share DIDs and VCs. An agent can access the wallet, which stores personal information about the user, to perform cryptographic operations on his behalf. An agent can be an edge agent, if it is hosted on the user's device, or a cloud agent, if it is hosted in the cloud.
- Credential Exchange Layer: this layer defines how credentials are managed.
- Governance Layer: this layer hosts the SGF [22] [23].

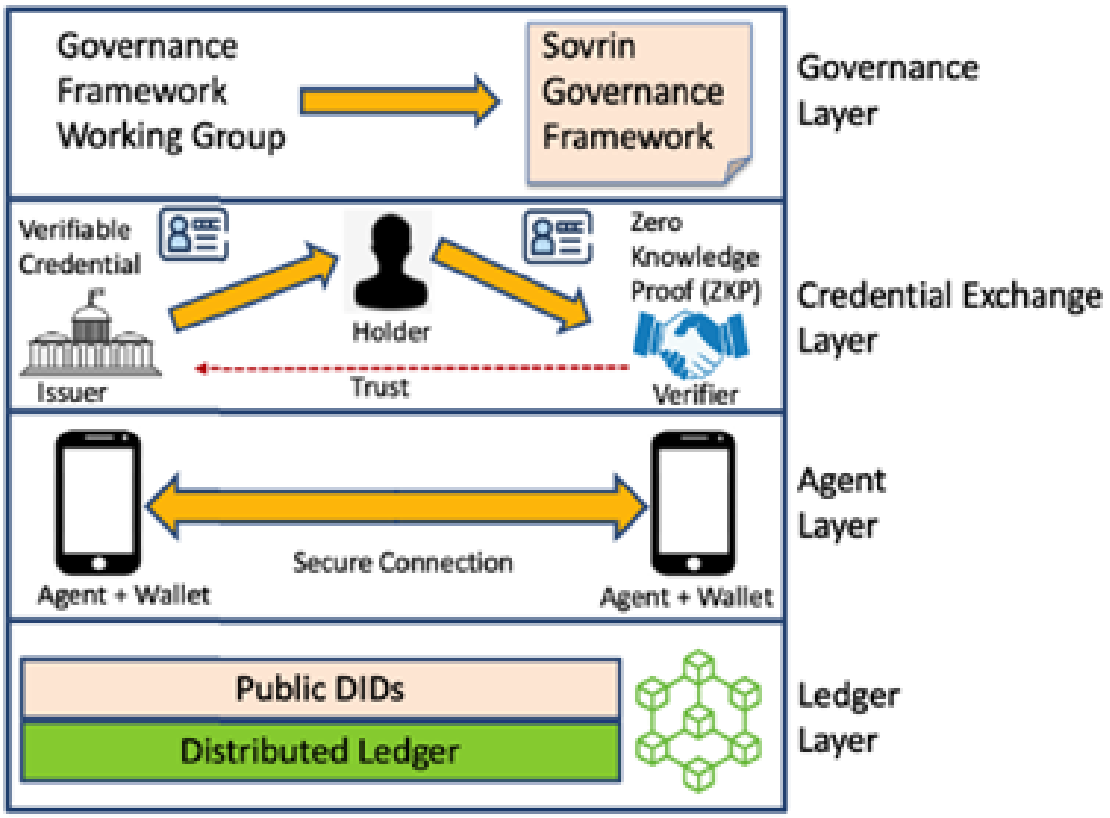

Figure 5.1. Sovrin layers: Governance Layer, Credential Exchange Layer, Agent Layer, Ledger Layer [22].

5.2 CredChain

CredChain is a Parity-based SSI platform that allows to securely manage users' credentials using a digital wallet. By using a blockchain DApp, a user can ask for a credential from a registered issuer and store it in his digital wallet. When sharing a credential, a user can remove some parts from it to minimize the oversharing problem. This is done by using a cryptographic primitive called "redactable signature" that allows him to sign only a part of the information in the credential. Moreover, it is possible to set up a time constraint to limit the period in which the verifier can have access to the shared information.

The CredChain architecture is composed of two parts: the SSI platform and a decentralized application layer that allows the user to interact with the blockchain network via a client wallet application. An issuer can use two storages: the Identity and Fact Data Repository and the Verifiable Credential. The first is used to store users' identity data and facts, while the second one is the cloud storage used to manage generated credentials. The SSI platform part contains a data layer and a service layer including all the necessary operations to manage identities. In particular, the Credential Services are divided into Selective Disclosure Authority (SDA), which provides the redactable signature service, and Verification. It is important to highlight that other SDA and Verification components can be added or replaced to use another selective disclosure scheme [48].

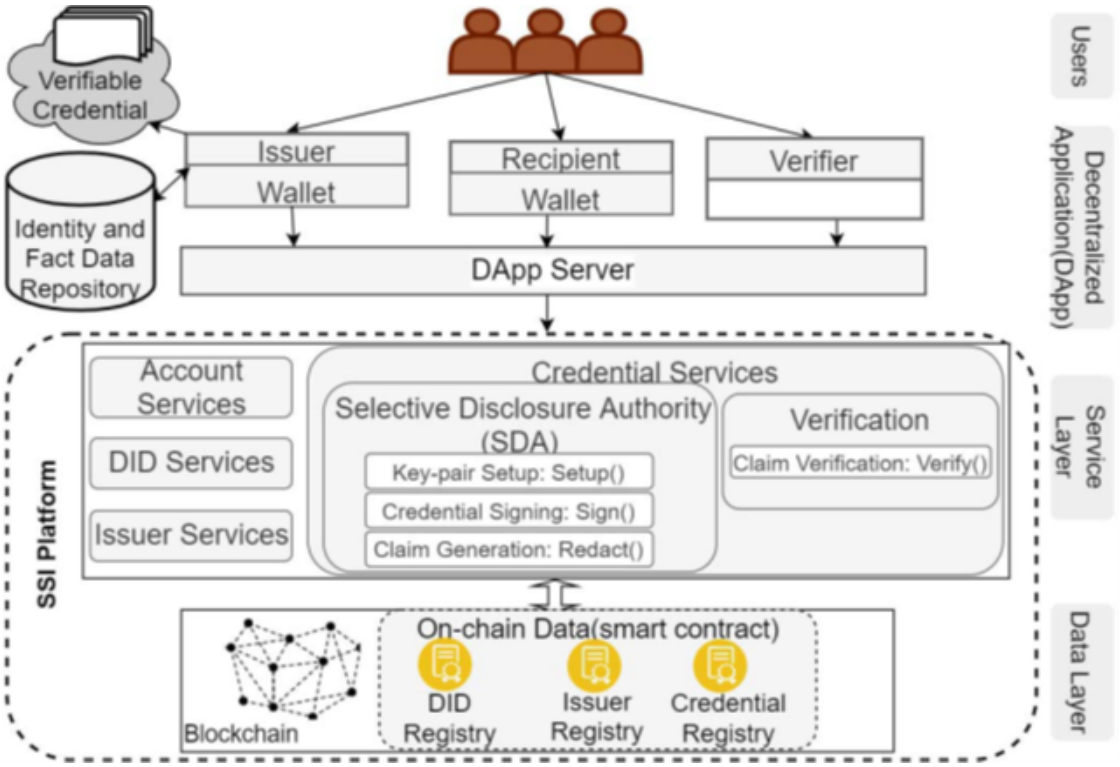

Figure 5.2. CredChain architecture [48].

5.3 Shocard

ShoCard is a mobile identity platform using the Bitcoin blockchain as a timestamping service. It has a fixed central server that manages the exchange of encrypted identity information between a user and a relying party. It integrates with existing SSO services and leverages on protocols like SAML and OpenID Connect [49].

The main peculiarity of this approach is in the Bootstrapping phase, in which the information about the user is scanned from a physical card by the camera of the mobile device. This information is then stored encrypted in the device together with a photo of the user. The main disadvantage of this approach is that Bitcoin transactions take 10 minutes to be mined into the blockchain and, moreover, you should wait for six more blocks to be mined before considering correct a transaction, and this could create challenges [50] [51].

5.4 Casper

Casper is a Turing-complete smart-contracting platform, which means that it can understand and execute any smart contract without any human interaction. It is supported by a Proof-of-Stake (PoS) and the consensus protocol of the network is called “Highway”.

The identities are stored in user’s mobile application, while the proof of these identities is stored on the blockchain. To verify the identity information, the ZKP mechanism is used to

associate the user credentials on the digital wallets and the credential proof on the blockchain [24] [52] [53] [54].

Casper uses a group of validators to verify transactions, and for participating in the consensus mechanism, the validators receive a token on the Casper Network called CSPR [53].

The Casper platform is composed of four layers:

- Distributed ledger: in this layer DIDs proofs and user consents are stored. The consents are needed because in Casper credential owners must provide different entities in the platform the permission to access their credentials.
- Smart contract layer: in this layer, the self-sovereign identity functions (as notification and identity/permission handling) are implemented by the smart contracts.
- Credential layer: this layer is used to create and exchange credentials for verification. Casper distinguishes two kinds of users: credential owners and admins, with the last category that includes credential approvers and credential verifiers. For these two kinds of users, two applications have been developed: identity owners use the "Casper mobile wallet application", while admins use the "Trace web application".
- Peer-To-Peer communication layer: the peer-to-peer exchange of data between credential owners and admins is performed in this layer. To verify or approve some credentials, an admin asks for permission to access those credentials from the owner, which by giving his consent transfers the required data and the cryptographic proof in the blockchain. This data is taken by the Casper mobile app from the local storage and sent to the Trace mobile application of the admin [24], [54].

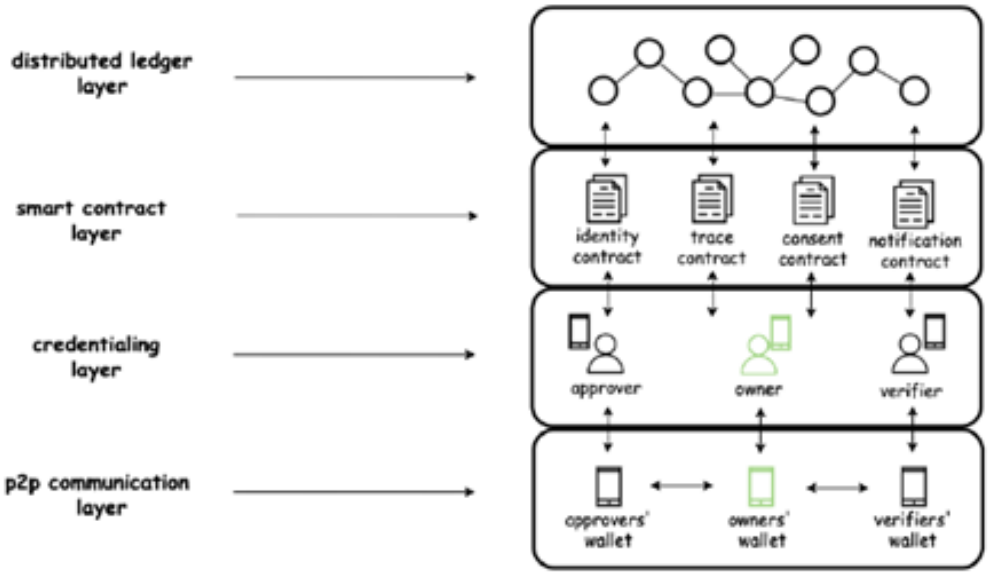

Figure 5.3. Casper architecture: distributed ledger layer, smart contract layer, credentialing layer, p2p communication layer [54].

Chapter 6

Implementation of ValID, a digital wallet based on the SSI model

To reach the purpose of this thesis, three modules have been created:

- *Wallet_web_app*: a web application representing a demo of a digital wallet.
- *Verifier*: an express server impersonating a verifier.
- *IdentityProvider*: an express server impersonating both an identity provider and an attribute provider.

Fig. 6.1 explains the relationship among the modules.

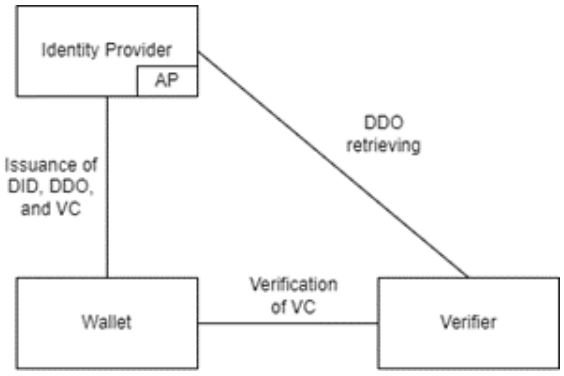

Figure 6.1. ValID modules.

The Identity Provider communicates with the wallet to issue DIDs, DID Documents, and Verifiable Credentials, and communicate with the Verifier to give him the DID Document related to a certain alias. The wallet communicates with the Verifier to verify the status of a credential. The communication among them uses HTTPS, using self-signed certificates for the IdentityProvider and the Verifier. This has been done to achieve the HTTPS advantages, including confidentiality: in this way, the personal information about the user is encrypted, so there is protection for the

data in transit. Since certificates are self-signed, the communication is not considered secure. As a future improvement of this project, real certificates can be used.

The ledger used in this project is the IOTA mainnet. This choice has been done to overcome the blockchain limitations described in chapter 4.2.

6.1 Wallet

Wallet_web_app is a React web application designed to run in a browser. It gives the possibility to the user to request and obtain a DID and a VC and to verify that the VC is valid. The project contains two main components: *Onboarding* and *GetCred*.

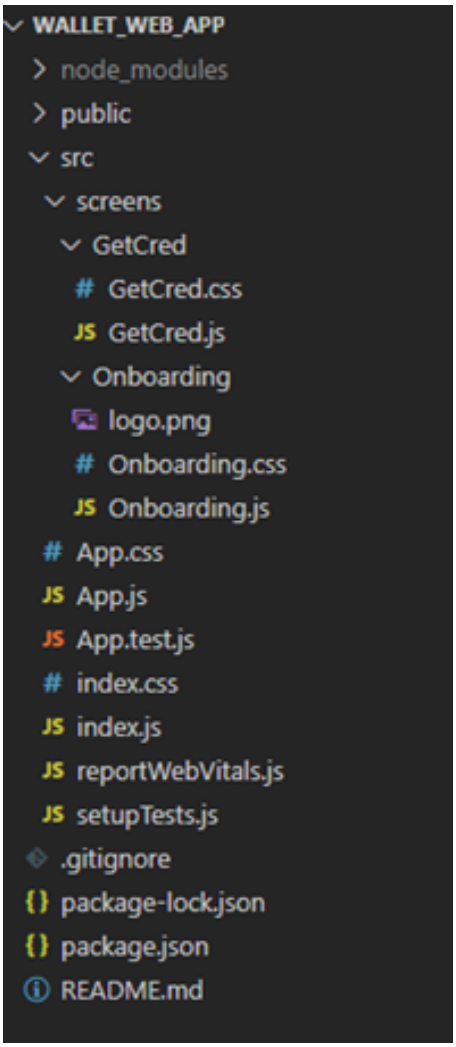

Figure 6.2. Wallet_web_app project structure.

The navigation between the screens is possible thanks to the "BrowserRouter" component of the *react-router-dom* library. It has been inserted in the "App" component, which has been

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { Provider as AlertProvider } from 'react-alert'
import AlertTemplate from 'react-alert-template-basic'
import App from './App';

const root = ReactDOM.createRoot(
    document.getElementById('root')
);
root.render(
    <React.StrictMode>
        <AlertProvider template={AlertTemplate}>
            <App/>
        </AlertProvider>
    </React.StrictMode>
);
```

Figure 6.3. File index.js inside Wallet_web_app.

inserted in the *index.js* file, the entry point of the project. The "AlertProvider" tag, imported from the *react-alert* library, is used to create alerts useful to make the user understand if he is doing something wrong.

In the App component, two variables are defined using the *useState* hook:

- *Alias*: it contains the alias chosen by the user;
- *Credential*: it contains the received VC.

```
import React, {useState} from 'react';
import {BrowserRouter, Routes, Route} from 'react-router-dom';
import Onboarding from './screens/Onboarding/Onboarding';
import GetCred from './screens/GetCred/GetCred';

const App = () => {
  const [alias, setAlias] = useState();
  const [credential, setCredential] = useState('');

  return (
      <React.Fragment>
        <BrowserRouter>
          <>
            <Routes>
              <Route
                  path="/"
                  element={
                    <Onboarding setAlias={setAlias} />
                  }
              />

              <Route
                  path="/getCred"
                  element={
                    <>
                      <GetCred alias={alias}
                              setCredential={setCredential}
                                  credential={credential}/>
                    </>
                  }
              />
            </Routes>
          </>
        </BrowserRouter>
      </React.Fragment>
  );
};
export default App;
```

Figure 6.4. App component inside Wallet_web_app.

These variables are passed to the components allowing them to modify or use their content. In particular:

- The Onboarding component needs to set the content of the alias variable after the generation of the DID of the user.
- The GetCred component needs to send the user alias to the Identity Provider to obtain a

verifiable credential, set the value of the credential variable using the setCredential function, and consult the value of the credential variable to select which components have to be shown on the screen and for verification purposes.

6.1.1 Onboarding

The Onboarding screen is composed of a title, an image, an input field, and a button. To obtain a DID, the user must insert an alias and click the *Register* button. If the user does not insert an alias, an alert will be shown on the screen, as represented in Fig. 6.7.

After that the user has inserted the alias and clicked the button, the page changes this content to explain the user why he has to wait for the DID.

If the alias has been used in the context of this application, the user must insert a new alias and a message will appear to explain to him what went wrong. If instead, the alias is available, the user is provided with a DID that is displayed on the screen.

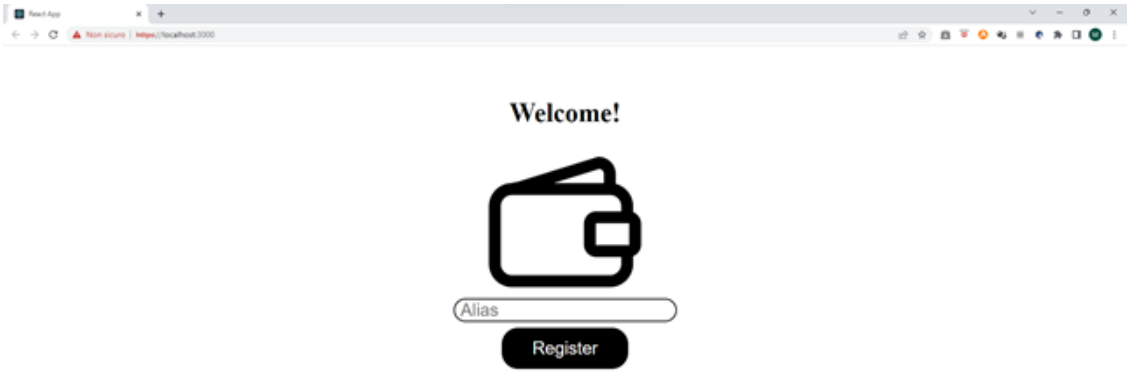

Figure 6.5. Onboading screen.

By clicking on the *Get a new VC!* button, the user is redirected to the GetCred screen. To take a trace of the alias inserted by the user, of the received DID, and of the changes in the page, four *useState* hooks are used. To perform the navigation between the three main pages of the application, the *useNavigate* function of the *react-router* library is used.

When the user inserts the alias and clicks on the *Register* button, the *handleRegistration* function is invoked.

The *handleRegistration* async function starts by checking if the user has inserted an alias. If the user has not done it before clicking on the *Register* button, an alert will be shown (Fig. 6.7), otherwise, the *verifying* variable is set as true to cause a change in the page, which will become as in Fig 6.8, and a *POST* request is performed. The request is performed by using the *axios*

Figure 6.6. Example of an Onboarding screen filled with an alias.

Figure 6.7. Alert of the Onboarding screen appearing after the pressure of the button without having inserted an alias.

library, his destination is the Identity Provider, and the data transported is the alias inserted by the user. When the response is received, if the content is *alias_used* the alias chosen by the user is not available, the error variable is set to true, and the screen will be as in Fig. 6.9. If, instead, the alias is available, the alias variable is set with the alias inserted by the user, the did variable is set with the response given by the Identity Provider, and the screen will become as in Fig. 6.10. In both cases, the *verifying* variable is set to false to allow the correct changes on the screen.

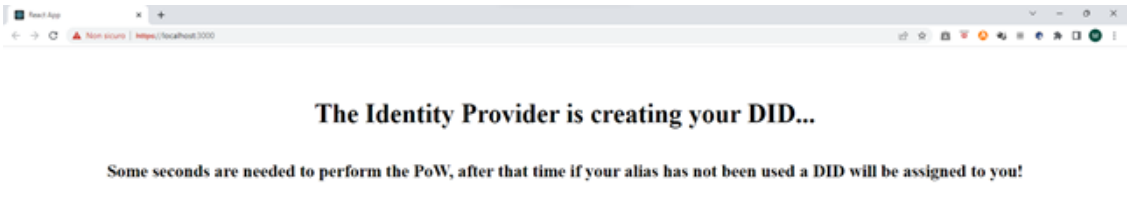

Figure 6.8. Screen appearing when the user is waiting for the DID.

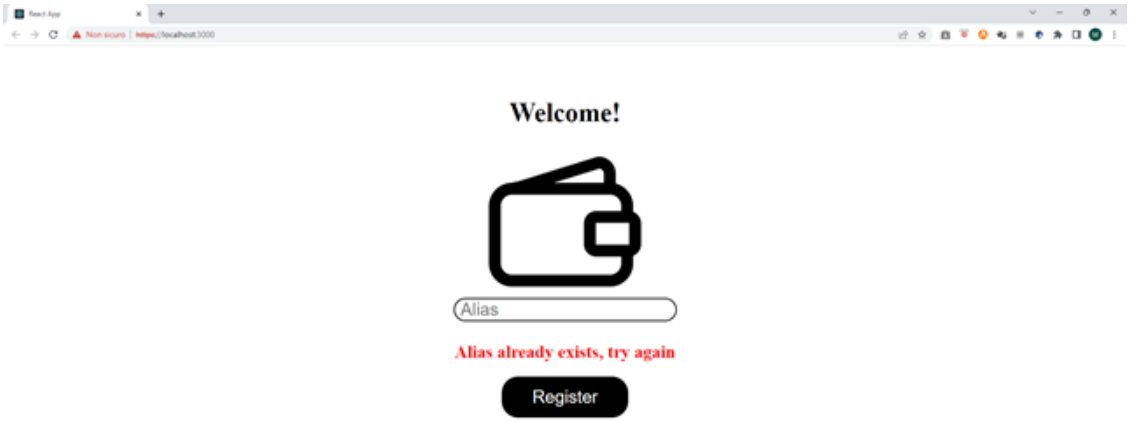

Figure 6.9. Error string appearing when the chosen alias has already been used.

To show the error on the screen, the piece of code in Fig. 6.13 is used. In this way, if the *error* variable has been set, the text will be visible, otherwise, it will not. The *verifying* variable is used, instead, together with the *did* one, to render the full screen, as shown in Fig. 6.14.

When *verifying* is true, the wallet is waiting for the DID from the Identity Provider. Before

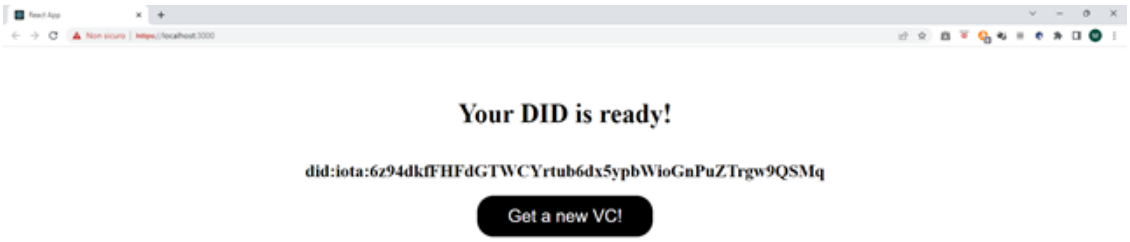

Figure 6.10. Screen showing the DID assigned to the user.

```
const [user, setUser] = useState('');
    const [did, setDid] = useState('');
    const [error, setError] = useState(false);
    const [verifying, setVerifying] = useState(false);
    const alert = useAlert()
    const navigate = useNavigate();

    function handleClick() {
        navigate("/getCred");
    }
```

Figure 6.11. Extract from Onboarding.js.

returning the DID, the IdP must communicate with the IOTA mainnet and perform a transaction to store DID and DID Document. This requires time, and for that reason the screen changes communicating it to the user.

```
async function handleRegistration() {
       if (user === '') {
           alert.show('Fill all the fields!');
       }
       else {
           setVerifying(true);
           const userDid = await axios.post(
               "https://127.0.0.1:8443/createIdentity",
               {
                   headers: {
                       "Content-Type": "application/json",
                   },
                   data: {
                       alias: user,
                   }
               }
           );

           if (userDid.data==="alias_used") {
               setError(() => true);
               setVerifying(false);
           }
           else {
              setAlias(() => user);
              setDid(() => userDid.data);
              setVerifying(false);
           }
       }
}
```

Figure 6.12. handleRegistration function.

```
{error===true ? (
    <h2 className="error">Alias already exists, try again</h2>)
    :
    (<></>)
}
```

Figure 6.13. Code for the appearance of the error string.

```
return (
    <React.Fragment>
        {did===''&&verifying===false?(
            <div className= "viewStyle">
            <h1 className='title'>Welcome!</h1>
            <img
                className="tinyLogo"
                src={logo}
                alt={"Logo"}/>
            <div>
                <input
                    onChange={(e) => setUser(e.target.value)}
                    placeholder="Alias"
                    className="inputField"
                />
            </div>
            {error===true ? (
                <h2 className="error">Alias already exists, try again</h2>)
                :
                (<></>)
            }
            <button
                onClick={handleRegistration}
                className="login_button"
            >Register</button>
        </div>
        )
    : verifying? (
        <div className= "viewStyle">
            <h1 className='title'>The Identity Provider is creating your
                DID...</h1>
            <h2 className='description'>
                Some seconds are needed to perform the PoW, after that time if
                    your alias has not been used a DID will be assigned to you!
            </h2>
        </div>
    ):(
        <div className= "viewStyle">
            <h1 className='title'>Your DID is ready!</h1>
            <h2 className='description'>{did}</h2>
            <button className="login_button" onClick={handleClick}>Get a new
                VC!</button>
        </div>
    )}
    </React.Fragment>
);
```

Figure 6.14. Return value of the Onboarding component.

6.1.2 GetCred

The GetCred screen is composed of a title, a description, three input fields, and a button. In this project, the verifiable credential to ask is related to the university attended by the user. He must insert his name, his surname, and his university, and by clicking on the button *Get VC* the VC will be generated.

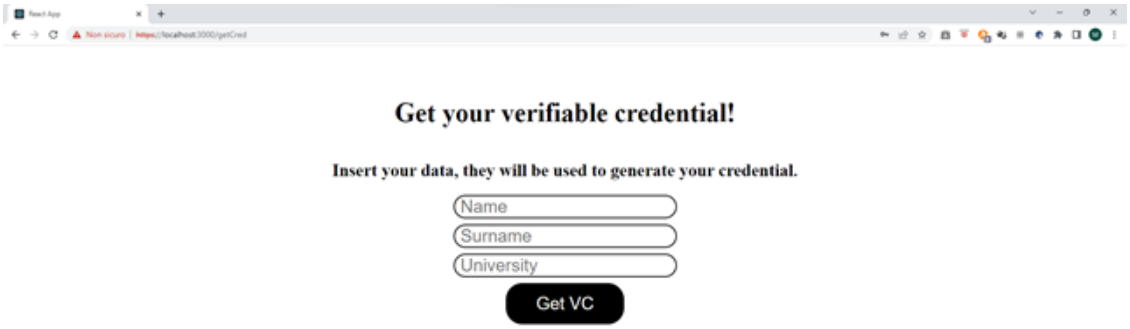

Figure 6.15. GetCred screen.

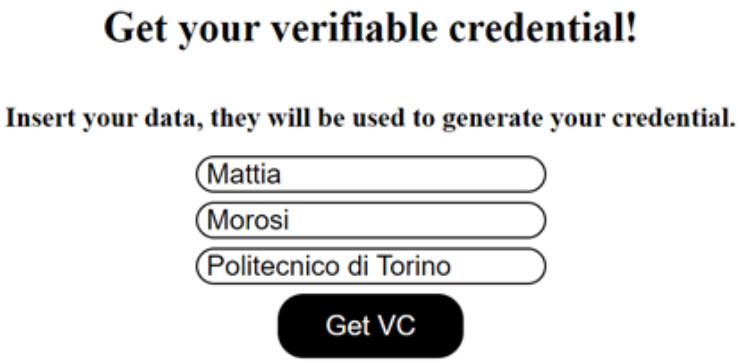

Figure 6.16. GetCred screen with filled fields.

If the user does not fill all the fields needed to ask for a credential, an alert will appear on the screen, as shown in Fig. 6.17.

Once the verifiable credential is generated, the wallet application displays it on the screen, as shown in Fig. 6.18.

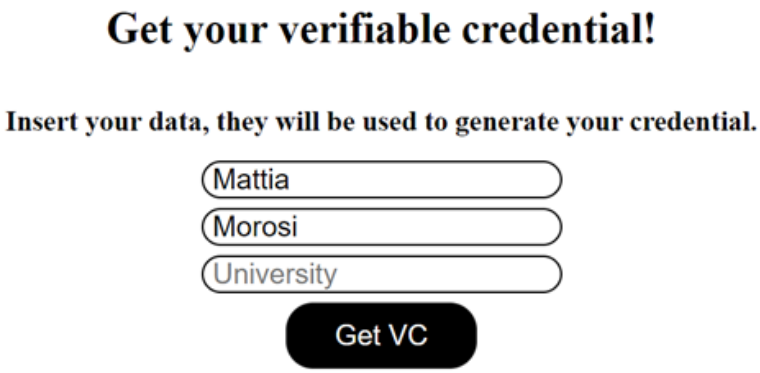

Figure 6.17. Alert shown when at least one of the fields are empty after the pressure of the button Get VC.

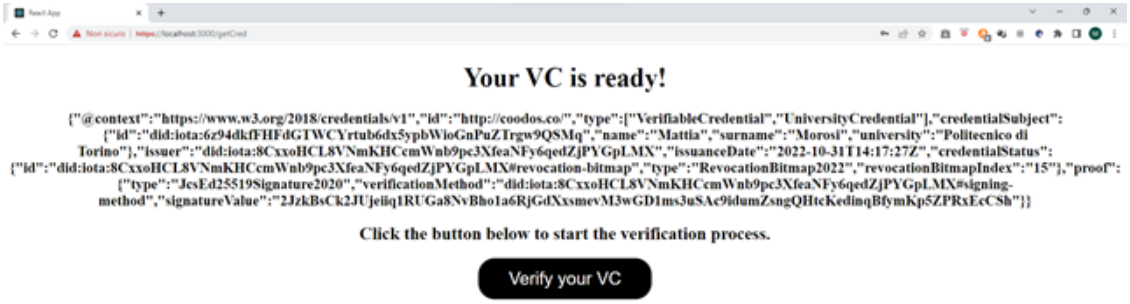

Figure 6.18. Screen showing the received VC.

By analysing the verifiable credential, it is noticeable that the DID field of the *credentialSubject* field is the same DID returned to the user in the Onboarding screen. Moreover, other useful information like the *DID* of the issuer, the *issuanceDate*, and the credential's *proof* can be identified.

By clicking on the *Verify your VC* button, the page changes showing the state of the Verifiable Credential. If the Verifiable Credential has not been revoked, the page will show a text saying *The VC is valid!*, as shown in Fig. 6.19. If, instead, the Verifiable Credential has been revoked, the text shown on the screen will be *The Verifiable Credential has been revoked!*, as shown in Fig. 6.20.

If there have been problems in retrieving the DID Document of the issuer to perform the verification, the text shown by the page will be *It is not possible to retrieve the DID Document of the issuer*, as shown in Fig. 6.21.

Figure 6.19. Screen showing the positive result of the verification of the VC.

The three fields that the user needs to fill to obtain the credential are managed through three variables generated using the *useState* hook, as can be noted in Fig. 6.22. Their value is updated each time the user changes the text of the related input fields. The fourth variable *result* will be used to contain the result of the verification process.

By clicking on the *Get VC* button, the *handleVCgeneration* function in Fig. 6.23 is invoked.

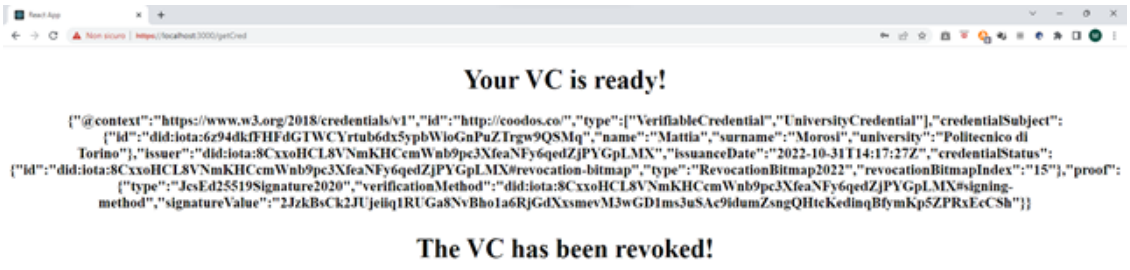

Figure 6.20. Screen showing the negative result of the verification of the VC.

Figure 6.21. Screen showed when an error occurs in the retrievement of the DID Document of the issuer of the VC.

Inside the *handleVCgeneration* function, a check on the fields is performed. If not all the fields are filled, an alert is shown, otherwise a *POST* to the Identity Provider is performed, including in the body the information inserted from the user. When the Identity Provider returns the Verifiable Credential, it is saved in the *credential* variable passed as a prop to the *GetCred* component.

```
const [name, setName] = useState('');
const [surname, setSurname] = useState('');
const [university, setUniversity] = useState('');
const [result, setResult] = useState('');
const alert = useAlert();
```

Figure 6.22. Extract from GetCred.js.

```
async function handleVCgeneration() {
    if (
        name === '' ||
        surname === '' ||
        university === ''
    ) {
        alert.show('Fill all the fields!');
    }
    else {
        const vc = await axios.post(
            "https://127.0.0.1:8443/createUserVC",
            {
                headers: {
                    "Content-Type": "application/json",
                },
                data: {
                    name: name,
                    surname: surname,
                    university: university,
                    userAlias: alias,
                }
            }
        );
        setCredential(() => vc.data)
    }
}
```

Figure 6.23. handleVCgeneration function.

The *credential* variable is used to show different elements on the screen depending on the fact that the Verifiable Credential has been received or not. If the Verifiable Credential has not been received, the input fields to capture user information are shown, otherwise the credential is shown on the screen with a button *Verify your VC*. The click on that button invokes the *verify* async function, which sends a *POST* request to the *Verifier* containing the Verifiable Credential to verify. Once the result of the verification is available, the *result* variable is set, and it is shown on the screen.

```
return (
    <React.Fragment>
        {credential===''?(
            <div className= "viewStyle">
                <h1 className='title'>Get your verifiable credential!</h1>
                <h2 className='description'>Insert your data, they will be
                    used to generate your credential.</h2>
                <div className='inputContainer'>
                    <input
                        onChange={(e) => setName(e.target.value)}
                        placeholder="Name"
                        className="inputField"
                    />
                    <input
                        onChange={(e) => setSurname(e.target.value)}
                        placeholder="Surname"
                        className="inputField"
                    />
                    <input
                        onChange={(e) => setUniversity(e.target.value)}
                        placeholder="University"
                        className="inputField"
                    />
                </div>
                <button
                    onClick={handleVCgeneration}
                    className="login_button"
                >Get VC</button>
            </div>)
        : (<div className='cred_div'>
                <h1 className='title'>Your VC is ready!</h1>
                <h3 className='VC'>{JSON.stringify(credential)}</h3>
                {!(result===''') ? (
                    <h2 className='title'>{JSON.stringify(result).slice(1,
                        -1)}</h2>
                ):(
                    <>
                        <h2 className='description'>Click the button below to
                            start the verification process.</h2>
                        <button onClick={verify}
                            className="verify_button">Verify your VC</button>
                    </>
                )}
            </div>)}
    </React.Fragment>
);
```

Figure 6.24. return value of the GetCred component.

```
async function verify() {
    const verification_result = await axios.post(
        "https://127.0.0.1:8444/verifyVC",
        {
            headers: {
                "Content-Type": "application/json",
            },
            data: {
                vc:credential
            }
        }
    );
    setResult(() => verification_result.data)
}
```

Figure 6.25. verify function.

6.2 Identity Provider

The identity provider is a server exposing endpoints to manage DIDs and Verifiable Credentials. It is based on express and uses HTTPS as a communication protocol.

To use HTTPS, a private key and a certificate are needed, and to generate them the OpenSSL library has been used. With the following instruction:

openssl genrsa -out filename.key 2048

An RSA private key of 2048 bits is generated and stored in a file called *filename.key*. Using this key, a *Certificate Signing Request (CSR)* can be created by writing on the terminal the following instruction:

openssl req -new -key filename.key -out filename.csr

In this instruction it is possible to identify a set of options:

- *req*: it is used to create a certificate request in PKCS#10 format.
- *-new*: it generates a new certificate request.
- *-key*: it specifies the name of the file containing the private key.
- *-out*: it specifies the name of the file that will contain the request.

Then, a self-signed certificate can be created by executing the following line of code:

openssl x509 -in filename.csr -out filename.crt -req -signkey filename.key -days 365

In this line of code, a set of options can be identified:

- *-in*: it specifies the name of the file to read.
- *-out*: it specifies the name of the file that will contain the certificate.
- *-req*: it specifies that among the arguments passed to the command there is a CSR.
- *-sign*: it specifies the key to use for generating the signature.
- *-days*: it specifies the validity period of the certificate [55] [56].

The structure of the project is reported in Fig. 6.26. In the *HTTPS* folder, the three files created with the previous instructions are contained. The structure contains also four files containing the functions of the Identity Provider (*createAP.js, createIdentity.js, createUserVc.js,* and *getDDO.js*), and the entry point of the project (*index.js*), which contains all the endpoints exposed by the Identity Provider.

In index.js, the key and the certificate are imported into the code by writing the lines in Fig. 6.27. Using the *fs* library, the files containing them are read and their value is saved in the variables *privateKey* and *certificate*. These variables are then inserted in the *credentials* variable, which will be later passed to the *createServer* function to create the HTTPS server.

To give the possibility to the *wallet_web_app* project to send HTTPS requests to the Identity Provider, the *Cross-Origin Resource Sharing (CORS)* mechanism is imported from the *cors* library and used (Fig. 6.28).

The set of endpoints exposed by the Identity Provider is reported in Fig. 6.29.

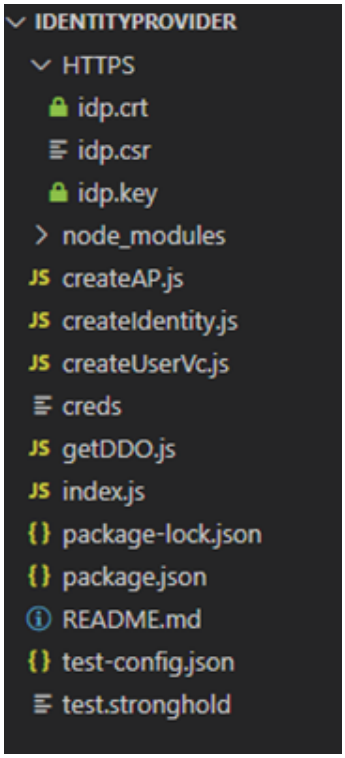

Figure 6.26. IdentityProvider project structure.

```
var privateKey = fs.readFileSync('./HTTPS/idp.key', 'utf8');
var certificate = fs.readFileSync('./HTTPS/idp.crt', 'utf8');
var credentials = {key: privateKey, cert: certificate};

const app = express();
app.use(express.json());
```

Figure 6.27. Retrievement of the private key and of the certificate of the Identity Provider.

```
app.use(
  cors({
    origin: ["https://localhost", "https://localhost:3000"],
    credentials: true,
    methods: ["GET", "POST", "PUT", "PATCH", "DELETE", "OPTIONS"],
    allowedHeaders: [
      "Origin",
      "X-Requested-With",
      "Content-Type",
      "Accept",
      "token",
      "Authorization",
    ],
  })
);
```

Figure 6.28. Usage of the CORS library to specify the allowed origins, methods, and headers.

```
app.post("/createIdentity", async function(req,res) {
    const { alias } = req.body.data;
    console.log("POST /createIdentity received, alias: "+alias)
    const did = await createIdentity(alias);
    if (did==="alias_used") res.send("alias_used");
    else res.json(did);
})

app.post("/getDDO", async function(req,res) {
  const { alias } = req.body.data;
  console.log("POST /getDDO received, alias: "+alias)
  const ddo = await getDDO(alias);
  if (ddo==="Not_found") res.send("Not_found");
  else res.json(ddo);
})


app.post("/createAP", async function(req,res) {
  const { alias } = req.body.data;
  console.log("POST /createAP received, alias: "+alias)
  const did = await createAP(alias);
  if (did==="alias_used") res.send("alias_used");
  else res.json(did);
})

app.post("/createUserVC", async function(req,res) {
  console.log("POST /createUserVC received");
  const { name, surname, university, userAlias } = req.body.data;
  console.log("Params: "+name+", "+surname+", "+university+", "+userAlias);
  const vc = await createUserVc(name, surname, university, userAlias);
  res.json(vc);
})
```

Figure 6.29. Identity Provider endpoints.

```
const port = 8443;
var httpsServer = https.createServer(credentials, app);
httpsServer.listen(port,()=>{
  console.log('Identity Provider is listening at port \$\{port\}')
})
```

Figure 6.30. Server creation and "listen" function invocation.

The exposed endpoints are the following:

- *createIdentity*: this endpoint is used to generate a DID for the user. The alias is collected from the body of the request, and it is passed to the *createIdentity* async function. When this function returns, if the alias is available the DID is returned to the user, otherwise the *alias_used* string is returned.
- *getDDO*: this endpoint is used to return the DID Document linked to an alias received in the body of the request. The *getDDO* function is called and, depending on the result, the DID document or the *Not_found* string is returned to the sender of the request.
- *createAP*: this endpoint is used to create an Attribute Provider. In the context of this project, it has been used only once to create an Attribute Provider that has been merged into the Identity Provider. Now, this endpoint is never used, but it is still part of the project because of possible reuse in the future.
- *createUserVC*: this endpoint is used to create a Verifiable Credential for the user identified by the alias *alias* received in the body of the request together with its name, its surname, and the attended university. The *createUserVC* function is called with the received parameters and the result is sent back to the sender of the request.

After having defined the endpoints, the HTTPS server is created by passing the *app* and the *credentials* variables and it is called the *listen* function to make the server able to process requests coming from the clients.

6.2.1 CreateIdentity

For the creation of a DID on the IOTA mainnet, the *@tanglelabs/identity-manager* library is needed. In particular, the *IdentityManager* class is used to manage this information. DID Documents created by the Identity Manager with the alias *test* are saved locally in the *test-config.json* file and loaded on the IOTA mainnet. The seeds used to create the keys associated with the accounts are instead saved in *test.stronghold*, a file created by using the IOTA Stronghold runtime secrets manager. The path used to create the *test.stronghold* file is passed to the *newInstance* function together with a *password* used to encrypt the information inside the file. Then, using this Identity Manager it is possible to create the identity for the user with the *createDid* function, which returns an *IdentityAccount* object. Among the parameters passed to this function, there is a *store*, that specifies in this case that the credentials related to the DID will be saved using the *FsStorageDriver* in the cred file. The credentials stored in this file are symmetrically encrypted with the private key related to the DID of the subject of the credential, and the credentials should

```
const { IdentityManager, Types } = require("@tanglelabs/identity-manager");
const path = require("path");

const createIdentity = async (alias) => {
    const manager = await IdentityManager.newInstance({
        filepath: path.resolve(__dirname, "./"),
        password: "foo",
        managerAlias: "test",
    });
    try {
        const did = await manager.createDid({ // DID loaded on mainnet and
            PoW made locally
            alias: alias,
            store: {
                type: Types.Fs,
                options: { filepath: path.resolve(__dirname, "./creds") },
                    //file to save credentials
            },
        });
        return (did.getDid().toJSON());
    }
    catch (e) {
        return "alias_used";
    }
};


module.exports = {createIdentity};
```

Figure 6.31. createIdentity function.

be inserted manually inside it, this function just creates the file. In the context of this project, this file is not used, because the credential is stored only in the wallet of the user. If the alias has been already used in the context of this Identity Manager, an error is thrown and the *createIdentity* function returns the *alias_used* string. Otherwise, the DID is extracted from the *IdentityAccount* object by using the *getDid* function, it is converted into a JSON object and it is returned to the caller of the function.

6.2.2 GetDDO

The *getDDO* function is used to retrieve the DID Document which refers to an alias passed as input to the function. After having instantiated the same Identity Manager used for the DID Document creation, the *getIdentityByAlias* function of the *IdentityManager* class is used. If the alias has not been used in the context of this Identity Manager, an error is thrown, and the function returns the *Not_found* string. Otherwise, the function returns the *IdentityAccount* related to the alias, the DID Document is extracted from it by using the *getDocument* function, and it is returned to

```
const { IdentityManager } = require("@tanglelabs/identity-manager");
const path = require("path");

const getDDO = async (alias) => {

    const manager = await IdentityManager.newInstance({
        filepath: path.resolve(__dirname, "./"),
        password: "foo",
        managerAlias: "test",
    });

    // Issuer loading
    try {
        const identity = await manager.getIdentityByAlias(alias);
        return identity.getDocument();
    }
    catch(e) {
        return "Not_found"
    }
};


module.exports = {getDDO};
```

Figure 6.32. getDDO function.

the caller of the function.

```
const { IdentityManager, Types } = require("@tanglelabs/identity-manager");
const path = require("path");

const createAP = async (alias) => {
    const manager = await IdentityManager.newInstance({
        filepath: path.resolve(__dirname, "./"),
        password: "foo",
        managerAlias: "test",
    });
    try {
        const did = await manager.createDid({ // DID loaded on mainnet and
            PoW made locally
            alias: alias,
            store: {
                type: Types.Fs,
                options: { filepath: path.resolve(__dirname, "./creds") },
                    //file to save credentials
            },
        });
        await did.attachSigningMethod("#signing-method");
        return (did.getDid().toJSON());
    }
    catch (e) {
        return "alias_used";
    }
};


module.exports = {createAP};
```

Figure 6.33. createAP function.

6.2.3 CreateAP

The *createAP* function has practically the same code as the *createIdentity* function. The main difference is the *await did.attachSigningMethod("signing-method")* line. This is used to give the created identity the possibility to perform a signature over the credentials that it will create. However, this function is not used in the project, its code is reported because it is part of it being used to create the only Attribute Provider present in this project.

```
const { IdentityManager } = require("@tanglelabs/identity-manager");
const path = require("path");

const createUserVc = async (name, surname, university, userAlias) => {

    const manager = await IdentityManager.newInstance({
        filepath: path.resolve(__dirname, "./"),
        password: "foo",
        managerAlias: "test",
    });

    // Issuer loading
    const issuer = await manager.getIdentityByAlias("http://example-ap.com");
    const identity = await manager.getIdentityByAlias(userAlias);
    const signedVc = await issuer.credentials.create({
      keyIndex: 15,
      id: "http://coodos.co",
      type: "UniversityCredential",
      fragment: "#signing-method",
      recipientDid: identity.getDid(),
      body: {
        name: name,
        surname: surname,
        university: university,
      },
    });
    return signedVc.toJSON()
};


module.exports = {createUserVc};
```

Figure 6.34. createUserVC function.

6.2.4 CreateUserVC

This function is used to create a Verifiable Credential to the user identified by the alias *userAlias*. After the instantiation of the Identity Manager, the identities of the issuer and of the subject of the credential are loaded by using the *getIdentityByAlias* function of the Identity Manager. After that, the *create* function is used to create and sign a Verifiable Credential. In this implementation, the identity identified with *http://example-ap.com* as alias signs the credential that refers to *http://coodos.co* domain. The DID of the subject of the VC is inserted in the *recipientDid* parameter of the *create* function. Each credential should be issued ideally with a unique key in a collection of keys, and in this case, it is used the *BitmapCollection2022*. This is done to avoid having a signing method per each key: in this way, the signing method is linked to a collection of keys, identified by a *keyindex*. The advantage is in the revocation of a credential: it is possible to revoke a credential by just revoking the *keyindex* used to sign it.

6.3 Verifier

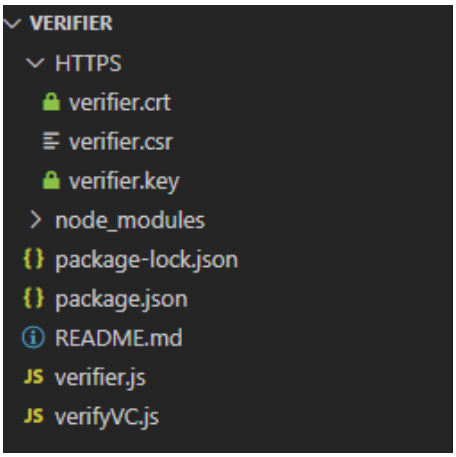

Figure 6.35. Verifier project structure.

The *Verifier* is a server exposing a single endpoint to verify Verifiable Credentials. It is based on *express* and uses *HTTPS* as a communication protocol. It has a very simple structure composed of an *HTTPS* folder containing the private key, the certification, and the Certificate Signing Request, the *verifier.js* file, which is the entry point of the project, and the *verifyVC.js* file, which contains the function to verify a Verifiable Credential.

By looking at the *verifier.js* file, it is noticeable that its configuration is similar to that of the Identity Provider, with just a different key, certificate, and port.

The only endpoint exposed by the verifier is */verifyVC*. It receives the Verifiable Credential in the body of the request, and it passes that credential to the *verifyVC* function. When this function returns, the status of the credential is returned. The *verifyVC.js* file contains the code of the *verifyVC* function.

```
var privateKey = fs.readFileSync('./HTTPS/verifier.key', 'utf8');
var certificate = fs.readFileSync('./HTTPS/verifier.crt', 'utf8');
var credentials = {key: privateKey, cert: certificate};

app.use(express.json());
app.use(
  cors({
    origin: ["https://localhost", "https://localhost:3000",
        "https://localhost:3001"],
    credentials: true,
    methods: ["GET", "POST", "PUT", "PATCH", "DELETE", "OPTIONS"],
    allowedHeaders: [
      "Origin",
      "X-Requested-With",
      "Content-Type",
      "Accept",
      "token",
      "Authorization",
    ],
  })
);
```

Figure 6.36. Retrievement of the private key and of the certificate of the verifier and usage of the CORS library to specify allowed origins, methods, and headers.

```
const port = 8444;
var httpsServer = https.createServer(credentials, app);
httpsServer.listen(port,()=>{
  console.log(Verifier is listening at port ${port})
})
```

Figure 6.37. Creation of the server and invocation of the "listen" function.

The first action performed in the *verifyVC* function is a *POST* request to the */getDDO* endpoint of the Identity Provider to retrieve the DID Document of the issuer of the Verifiable Credential. In the context of this project, the only entity that is able to issue credentials is *https://example-ap.com*, so it is possible to pass directly this alias. Otherwise, the alias of the issuer can be retrieved from the Verifiable Credential. If the search of the DID Document fails, the function returns *null*. Otherwise, the result of the *isCredentialValid* function is returned. This function is imported from the *@tanglelabs/identity-manager* library, it takes as input the Verifiable Credential and the DID Document of the issuer of that credential and checks both the signature on the credential and its state (if the credential has been revoked or not). Since the certificate of the Identity Provider is a self-signed one, it is needed to create a custom instance of the *axios* library specifying to do not reject connections not authorized from the list of supported Certification Authorities.

```
app.post("/verifyVC", async function(req,res) {
    console.log("POST /verifyVC received");
    const {vc} = req.body.data;
    const vr = await verifyVC(vc);
    if (vr === null) res.json("It is not possible to retrieve the DID
        Document of the issuer");
    else if (JSON.stringify(vr)==="false") res.json("The VC has been
        revoked!");
    else res.json("The VC is valid!");
})
```

Figure 6.38. /verifyVC endpoint.

```
const { isCredentialValid } = require("@tanglelabs/identity-manager");
const {Credential, Document} = require("@iota/identity-wasm/node")
const axios = require("axios");
const https=require("https");

const instance = axios.create({
    httpsAgent: new https.Agent({
      rejectUnauthorized: false
    })
});

const verifyVC = async (vc) => {
    const ddo = await instance.post(
        "https://127.0.0.1:8443/getDDO",
        {
            headers: {
                "Content-Type": "application/json",
            },
            data: {
                alias: "http://example-ap.com",
            }
        }
    );
    if (ddo.data === "Not found") return null;
    return await isCredentialValid(Credential.fromJSON(vc),
        Document.fromJSON(ddo.data));
};


module.exports = {verifyVC};
```

Figure 6.39. verifyVC function.

6.3.1 Diagrams

The flowchart of the project is reported in Fig. 6.40.

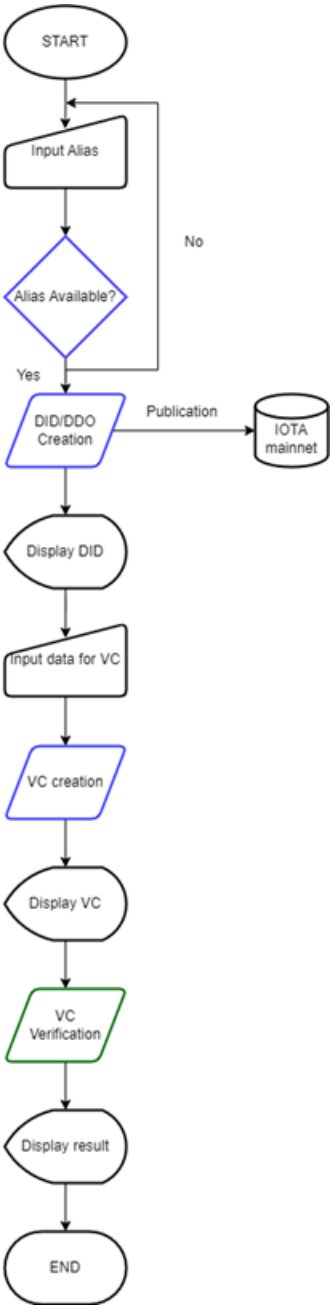

Figure 6.40. IdentityProvider project structure.

A scheme of the actions performed by each module is reported in Fig. 6.41. The colors inside the flowchart are related to the ones in the image below: the green blocks represent actions

performed by the Verifier, the blue blocks represent actions performed by the Identity Provider, and the black blocks represent actions performed by the Wallet.

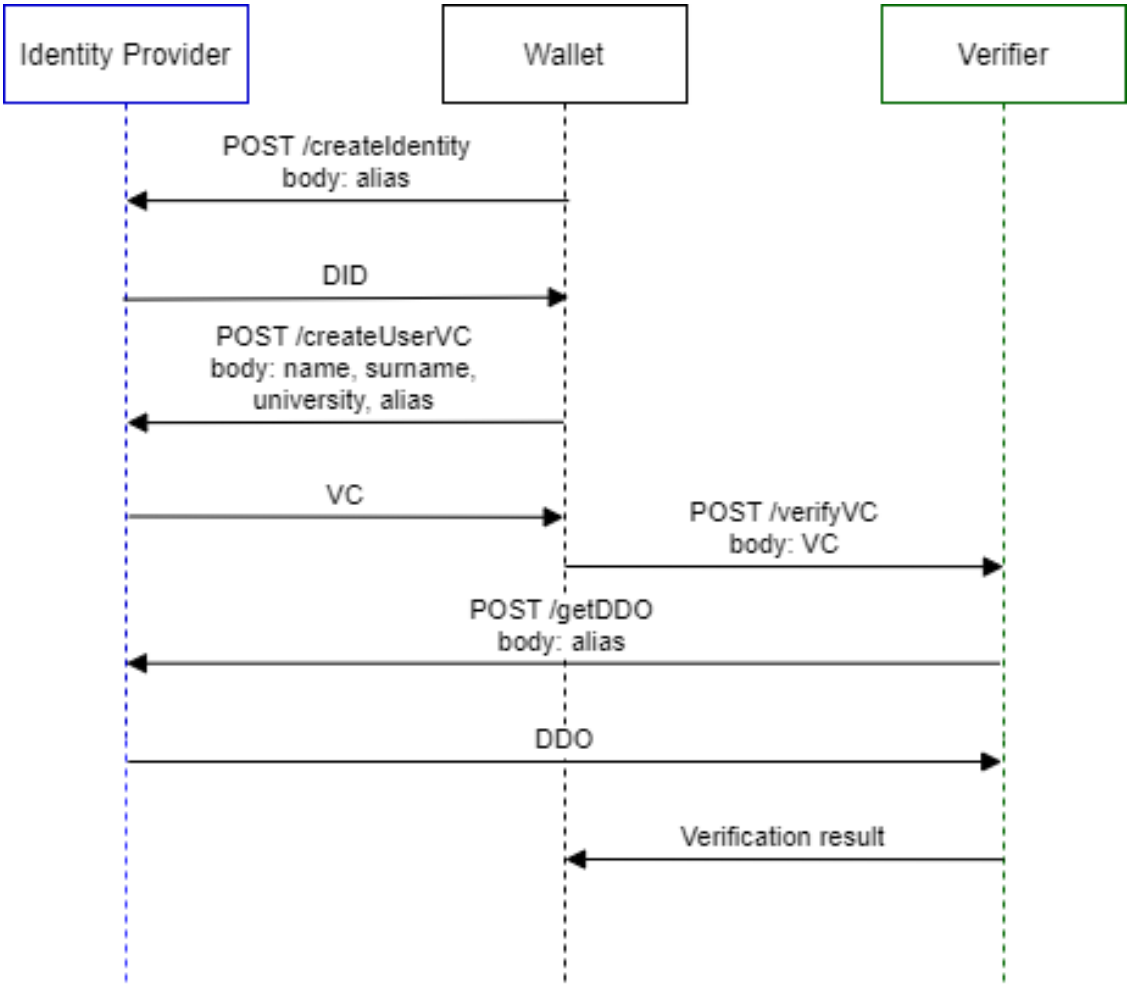

Figure 6.41. Interactions between ValID modules.

Chapter 7

Installation Guide

The first step is to extract the files contained in ValID.zip, then for the usage of each module a different procedure must be followed.

7.1 Wallet_web_app

7.1.1 Installation

Open a terminal in the *wallet_web_app* folder and write on the terminal *npm install* and *npm start*. This will run the app at *https://localhost:3000*.

To use the application, you have to run both the Verifier and the Identity Provider.

In the communication with the Verifier and the Identity Provider, HTTPS with self-signed certificates is used. To be able to use the wallet application, it is necessary to start the Verifier and the Identity provider, to open a route of each of them on the browser (e.g., *https://127.0.0.1:8443/createUserVC* for the Verifier and *https://127.0.0.1:8443/createIdentity* for the Identity Provider) and to consent the communication.

7.1.2 Usage

When the app starts, it is possible to insert an Alias to require the DID. By pressing the *Register* button, the DID generation starts. If the alias has not been used yet, the DID will appear on the screen, otherwise a new alias must be inserted. Once the DID is created, it is possible to request for a credential by pressing the *Get a new VC!* button. In the page that will be loaded, the fields of the credential to request must be inserted, and by pressing the *Get VC* button, the Verifiable Credential will be created and displayed. Then, it is possible to verify if the credential has been revoked by clicking on the *Verify your VC* button. After this, the result of the verification will be displayed on the screen.

7.2 Identity Provider

Open a terminal in the *IdentityProvider* folder and write on the terminal *node .\index.js*. This will run the app at *https://localhost:8443*.

7.3 Verifier

Open a terminal in the *Verifier* folder and write on the terminal *node .\verifier.js*. This will run the app at *https://localhost:8444*.

Chapter 8

Conclusion

The aim of this project was to build an architecture for the usage of a digital identity in a Self-Sovereign Identity (SSI) scenario. The SSI model empowers the user giving him full control over his personal information, without the need to rely on third parties for its management. During the last years, people have used digital identities to identify themselves online being unable to perform some activities in presence. In this context, each user is identified by a Decentralized Identifier and digital wallets are used to store verifiable credentials given by an issuer and to present them to a verifier. Issuer and verifier are linked from a trust relationship established by using a Distributed Ledger Technology (DLT). More DLTs can be used, but thanks to its scalability the IOTA mainnet has been used in this project. The underlying DLT is an important aspect to take into consideration when designing an SSI solution: the Tangle is the best choice in terms of speed and costs since transactions have no fees. On this DLT only public information useful for credential verification is stored, in particular DIDs and DID Documents, while personal information about the user like verifiable credentials is stored locally on the digital wallet. This gives the possibility to users to have full control over it and to decide when, with whom, and for which purpose to share their personal information, and this is one of the main advantages achievable by using this kind of solution.

ValID project is composed of three modules: a web application representing the demo of a digital wallet and two express servers representing the issuer (called Identity Provider) and the verifier. The wallet is a demo because this version does not store information between two different executions of the code. The task of the issuer is to provide the DID and the VC to the user and that of verifier is to check the validity of a verifiable credential presented by the user. Since personal information is transmitted between the entities, a secure channel among them has been created by using HTTPS to achieve server authentication, integrity, and data confidentiality. The only entity that interacts with the IOTA mainnet is the issuer which stores the DID Document. This requires an available internet connection and some seconds to perform a Proof of Work (PoW), which is not needed to reach a consensus as in blockchain solutions but just to avoid spam. The PoW can be done both locally or remotely by using an external device. In this project it is performed locally by the issuer when creating the DID for the user.

Since storing information on the user's personal device is an important task to be achieved, in a future release would be important to insert the possibility for the user to store it avoiding repeating the process every time he starts the application. Another possible improvement regards the certificates signature. Currently, to perform the HTTPS handshake, self-signed certificates are used. This can be acceptable in the first stage of the project, but in a future release two options might be taken into consideration. The first is to use not self-signed certificates; the

second regards using public keys inserted in the DID Documents on the IOTA mainnet to provide data confidentiality and mutual or single authentication by implementing a challenge-response protocol and a MIC to provide integrity. This last solution can be adopted to remove the usage of X.509 certificates and to be independent from certification authorities, relying only on public information available on the IOTA mainnet.

Finally, it could be inserted the possibility for the user to select which part of the credential to share with a verifier giving him even more control over his personal data.

Bibliography

[1] E. Bandara, X. Liang, P. Foytik, S. Shetty, and K. de Zoysa, "A blockchain and self-sovereign identity empowered digital identity platform", Proceedings - International Conference on Computer Communications and Networks, ICCCN, Manchester Metropolitan University (UK), Jul. 2021, DOI 10.1109/ICCCN52240.2021.9522184

[2] D. G. Berbecaru, A. Lioy, and C. Cameroni, "On enabling additional natural person and domain-specific attributes in the eidas network", IEEE Access, vol. 9, 2021, pp. 134096–134121, DOI 10.1109/ACCESS.2021.3115853

[3] N. Naik and P. Jenkins, "Your identity is yours: Take back control of your identity using gdpr compatible self-sovereign identity", Proceedings of 2020 7th IEEE International Conference on Behavioural and Social Computing, Bournemouth (UK), Nov. 2020, DOI 10.1109/BESC51023.2020.9348298

[4] I. A. Domingo, "SSI eIDAS Legal Report How eIDAS can legally support digital identity and trustworthy DLT-based transactions in the Digital Single Market", 2020

[5] R. Saracco, "An accelerated digital transformation, courtesy of the recent pandemic", 2020 ITU Kaleidoscope: Industry-Driven Digital Transformation, Dec. 2020, DOI 10.23919/ITUK50268.2020.9303181

[6] D. Pohn, M. Grabatin, and W. Hommel, "Eid and self-sovereign identity usage: An overview", Electronics, vol. 10, 2021, DOI 10.3390/electronics10222811

[7] N. Naik and P. Jenkins, "Self-sovereign identity specifications: Govern your identity through your digital wallet using blockchain technology", 2020 8th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, MobileCloud, Aug. 2020, pp. 90–95, DOI 10.1109/MobileCloud48802.2020.00021

[8] World Wide Web Consortium, `https://www.w3.org`

[9] D. Berbecaru and C. Cameroni, "Atema: An attribute enablement module for attribute retrieval and transfer through the eidas network", 2020 24th International Conference on System Theory, Control and Computing, ICSTCC 2020, Oct. 2020, pp. 532–539, DOI 10.1109/ICSTCC50638.2020.9259642

[10] A. Alonso, A. Pozo, J. Choque, G. Bueno, J. Salvachua, L. Diez, J. Marin, and P. L. Alonso, "An identity framework for providing access to fiware oauth 2.0-based services according to the eidas european regulation", IEEE Access, vol. 7, pp. 88435–88449

[11] European Commission, `https://ec.europa.eu/`

[12] D. Berbecaru, A. Lioy, and C. Cameroni, "Providing login and wi-fi access services with the eidas network: A practical approach", IEEE Access, vol. 8, 2020, pp. 126186 – 126200, DOI 10.1109/ACCESS.2020.3007998

[13] D. Berbecaru, A. Lioy, and C. Cameroni, "Electronic identification for universities: Building cross-border services based on the eidas infrastructure", Information, vol. 10, 2019, DOI 10.3390/info10060210

[14] D. Berbecaru, A. Lioy, and C. Cameroni Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, 2020

[15] Thales Group, https://www.thalesgroup.com
[16] General Data Protection Regulation, https://gdpr.eu/what-is-gdpr/
[17] G. Kondova and J. Erbguth, "Self-sovereign identity on public blockchains and the gdpr", ACM Symposium on Applied Computing, Mar. 2020, pp. 342–345, DOI 10.1145/3341105.3374066
[18] OASIS, http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html
[19] B. Podgorelec, L. Alber, and T. Zefferer, "What is a (digital) identity wallet? a systematic literature review", 2022 IEEE 46th Annual Computers, Software, and Applications Conference, COMPSAC 2022, 2022, pp. 809–818, DOI 10.1109/COMPSAC54236.2022.00131
[20] Evernym, https://www.evernym.com/
[21] Evernym, https://gitlab.com/evernym/mobile/connectme
[22] N. Naik and P. Jenkins, "Sovrin network for decentralized digital identity: Analysing a self-sovereign identity system based on distributed ledger technology", ISSE 2021 - 7th IEEE International Symposium on Systems Engineering, Sep. 2021, DOI 10.1109/ISSE51541.2021.9582551
[23] B. N. Eddine, A. Ouaddah, and A. Mezrioui, "Exploring blockchain-based self sovereign identity systems: Challenges and comparative analysis", 3rd Conference on Blockchain Research and Applications for Innovative Networks and Services, BRAINS 2021, Sep. 2021, pp. 21–22, DOI 10.1109/BRAINS52497.2021.9569821
[24] E. Bandara, X. Liang, P. Foytik, S. Shetty, C. Hall, D. Bowden, N. Ranasinghe, and K. D. Zoysa, "A blockchain empowered and privacy preserving digital contact tracing platform", Inf. Process. Manag., vol. 58, 2021, p. 102572, DOI 10.1016/j.ipm.2021.102572
[25] Jolocom, https://jolocom-lib.readthedocs.io/en/latest
[26] Jolocom, https://jolocom.io/
[27] Trinsic, https://trinsic.id
[28] Trinsic, https://docs.trinsic.id/docs
[29] Linus Naumann, https://medium.com/
[30] Tangle Labs, https://tanglelabs.io/vira
[31] IOTA, https://www.identity.tools/
[32] eID, https://www.electronicid.eu
[33] eIDAS Expert Group, https://cloud.eid.as/index.php/s/DQ5aRjyzJDNKXpW
[34] M. Belotti, N. BoÅ¾iÄ, G. Pujolle, and S. Secci, "A vademecum on blockchain technologies: When, which, and how", IEEE Communications Surveys and Tutorials, vol. 21, Oct. 2019, pp. 3796–3838, DOI 10.1109/COMST.2019.2928178
[35] G. Falazi, M. Hahn, U. Breitenbucher, F. Leymann, and V. Yussupov, "Process-based composition of permissioned and permissionless blockchain smart contracts", 2019 IEEE 23rd International Enterprise Distributed Object Computing Conference, EDOC 2019, Oct. 2019, pp. 77–87, DOI 10.1109/EDOC.2019.00019
[36] Investopedia, https://www.investopedia.com/terms/t/tangle-cryptocurrency.asp
[37] Tangle Bay, https://tanglebay.com
[38] Fiware, https://fiware-tutorials.readthedocs.io/en/latest/iot-over-iota-tangle.html
[39] IOTA, https://iota-beginners-guide.com/dlt/tangle
[40] Ethereum, https://ethereum.org/en/zero-knowledge-proofs
[41] A. Satybaldy and M. Nowostawski, "Review of techniques for privacy-preserving blockchain systems", BSCI 2020 - Proceedings of the 2nd ACM International Symposium on Blockchain and Secure Critical Infrastructure, Co-located with AsiaCCS 2020, Oct. 2020, pp. 1–9, DOI 10.1145/3384943.3409416
[42] Cem Dilmegani, https://research.aimultiple.com/zero-knowledge-proofs/

[43] Akash Takyar, https://www.leewayhertz.com/zero-knowledge-proof-and-blockchain
[44] Mangrovia, https://www.mangrovia.solutions/en/
[45] Rob Behnke, https://ethereum.org/en/zero-knowledge-proofs
[46] N. Naik and P. Jenkins, "Governing principles of self-sovereign identity applied to blockchain enabled privacy preserving identity management systems", ISSE 2020 - 6th IEEE International Symposium on Systems Engineering, Oct. 2020, DOI 10.1109/ISSE49799.2020.9272212
[47] N. Naik and P. Jenkins, "Does sovrin network offer sovereign identity?", ISSE 2021 - 7th IEEE International Symposium on Systems Engineering, Sept. 2021, DOI 10.1109/ISSE51541.2021.9582472
[48] R. Mukta, J. Martens, H. young Paik, Q. Lu, and S. S. Kanhere, "Blockchain-based verifiable credential sharing with selective disclosure", 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Dec. 2020, DOI 10.1109/TrustCom50675.2020.00128
[49] Arun Penmetsa, https://www.linkedin.com/
[50] M. D. E.-C. E. K. Samia El Haddouti, "Analysis of identity management systems using blockchain technology", 2019 International Conference on Advanced Communication Technologies and Networking (CommNet), Apr. 2019, DOI 10.1109/COMMNET.2019.8742375
[51] P. Dunphy and F. A. P. Petitcolas, "A first look at identity management schemes on the blockchain", IEEE Secur Priv, vol. 16, Jul. 2018, pp. 20–29, DOI 10.1109/MSP.2018.3111247
[52] CasperLabs, https://casperlabs.io/
[53] CasperLabs, https://docs.casperlabs.io/
[54] E. Bandara, S. Shetty, R. Mukkamala, X. Liang, P. Foytik, N. Ranasinghe, and K. D. Zoysa, "Casper: a blockchain-based system for efficient and secure customer credential verification", Journal of Banking and Financial Technology, Dec. 2021, DOI 10.1007/s42786-021-00036-3
[55] Manoj Srinivas, https://www.codespeedy.com/how-to-enable-https-in-express-js
[56] The OpenSSL project, http://www.openssl.org/